VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Klasifikace malware za použití ANN a příbuzných metod

Malware Cassification by Means of ANN and Similar Methods

2018 Marek Kajfosz

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Marek Kajfosz**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 1801T064 Informační a komunikační bezpečnost

Téma: Klasifikace malware za použití ANN a příbuzných metod
Malware Cassification by Means of ANN and Similar Methods

Jazyk vypracování: čeština

Zásady pro vypracování:

Práce je zaměřena na pochopení principů vybraných typů počítačových virů a jejich identifikaci pomocí umělé inteligence. Cílem a účelem práce je vytvořit sadu ukázkových případových studií vysvětlující analýzu a identifikaci viru didaktickým způsobem. Cílem je experimentálně naprogramovat ANN a provézt klasifikaci vzorků SW za účelem odhalení malware podle pokynů vedouciho DP.

Předpokládaná struktura práce je:
1. Seznámení se s problematikou.
2. Volba vhodného programovacího prostředí.
3. Volba vhodných algoritmů z oblasti technik ANN a evolucí.
4. Programová realizace těchto algoritmů.
5. Analýza a identifikace vybraných virů - již existujících vzorků.
6. Tvorba uživatelského manuálu.

Seznam doporučené odborné literatury:

[1] Merhaut F., Zelinka I., Úvod do počítačové bezpečnosti, Fakulta aplikované informatiky, UTB ve Zlíně, Zlín, 2009
[2] Peter Szor, Počítačové viry - analýza útoku a obrana, Zoner Press
[3] Zelinka I., Oplatková Z., Šeda M., Ošmera P., Včelař F., Evolutionary techniques – principles and applications, BEN, Prague, 2008, 598 p.
[4] Schmidhuber, Jürgen. "Deep learning in neural networks: An overview." Neural networks 61 (2015): 85-117.
[5] Singer, Peter W., and Allan Friedman. Cybersecurity: What Everyone Needs to Know. Oxford University Press, 2014.
[6] Moshchuk, Alexander, Tanya Bragin, Steven D. Gribble, and Henry M. Levy. "A Crawler-based Study of Spyware in the Web." In NDSS, vol. 1, p. 2. 2006. Harvard
[7] Balthrop, Justin, Stephanie Forrest, Mark EJ Newman, and Matthew M. Williamson. "Technological networks and the spread of computer viruses." Science 304, no. 5670 (2004): 527-529.
[8] Demuth, Howard B., Mark H. Beale, Orlando De Jess, and Martin T. Hagan. Neural network design. Martin Hagan, 2014.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **prof. Ing. Ivan Zelinka, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018 .....

Abstrakt

V dnešní době prudce narůstá množství nových počítačových hrozeb. To způsobuje, že metody manuální analýzy a detekce malware užitím signatur zaostávají. Díky zvyšujícímu se výpočetnímu výkonu a inovacím v oboru strojového učení, se neuronové sítě zdají být velmi slibnou alternativou. Účelem teoretické části této práce je definovat potřebné pojmy, principy a definice nutné k pochopení následující praktické části, která má za cíl naimplementovat a popsat klasifikaci malware ze zvoleného datasetu na základě jejich rodin.

Klíčová slova: klasifikace, konvoluční sítě, malware, neuronové sítě, strojové učení

Abstract

Nowadays, the number of computer threads is rising rapidly. This is making manual analysis and signature-based malware detection obsolete. Due to increase in computational power and improvements in the machine learning field, the usage of neural networks for analysis seems to be a very promising alternative. The purpose of theoretical section of this thesis is to define neccessary terms, principles and definitions in order to understand following practical part, which goal is to implement and describe malware classification based on malware families.

Key Words: classification, convolutional networks, machine learning, malware, neural networks

Obsah

Seznam použitých zkratek a symbolů

ANN	–	Artificial Neural Network - Umělá neuronová síť
ASCII	–	American Standard Code for Information Interchange
CNN	–	Convolutional Neural Network - Konvoluční neuronová síť
CSV	–	Comma Separated Values
FTP	–	File Transfer Protocol
GPU	–	Graphic Processing Unit - Grafický procesor
GRU	–	Gated Recurrent Unit
LSTM	–	Long Short Term Memory
MLP	–	Multi Layered Perceptron - Vícevrstvý perceptron
ReLU	–	Rectified Linear Unit
RNN	–	Recurrent Neural Network - Rekurentní neuronová síť
SGD	–	Stochastic Gradient Descent
SW	–	Software
XOR	–	eXclusive OR - exkluzivní disjunkce

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Útoky za pomocí škodlivého kódu jsou dnes na denním pořádku. V průměrů každý 131. email obsahuje určitým způsobem škodlivou přílohu, či odkaz. Dennodenně se přibližně 4 000 lidí stane obětí útoku, kdy útočník zablokuje přístup k zařízení napadeného a požaduje „výkupné“. Dle oficiálního shrnutí hrozeb vydaného společností McAfee v prosinci 2017[1] přibylo ve třetím čtvrtletí roku 2017 rekordních 57,6 miliónu nových vzorků potencionálně škodlivých programů. Pokud bychom počítali, že kvartál čítá 90 dní, pak se dostáváme na průměrných 600 000 nových souborů denně.

Veškeré tyto statistiky vedou k závěru, že kyberkriminalita je stále závažnějším problémem. Vzhledem k obrovskému množství přibývajících dat je prakticky nemožné vykonávat analýzu těchto útoků lidskými silami. Proto se v kyberbezpečnostním průmyslu začaly vyvíjet nové formy obrany prostřednictvím automatizace, respektive pomocí strojového učení. Umělé neuronové sítě jsou jedním z modelů, který je pro daný účel možno využít a vzhledem k nedávným pokrokům v tomto oboru, se zdá být velmi perspektivním nástrojem. Cílem této diplomové práce je seznámit čtenáře s problematikou malware a neuronových sítí, vyjmenovat některé základní i složitější struktury a popsat již existující řešení využívající zmíněných principů. Po této teoretické části práci následuje část experimentální, v níž dochází k implementaci extrakce několika sad informací ze zvoleného datasetu, na jejichž základě jsou prováděny klasifikace. Finálním cílem je pak vyhodnocení a srovnání těchto různých klasifikačních metod.

2 Terminolgie

Oba obory kterých se tato práce týká - tedy problematika strojového učení a malware, se rozvíjejí velmi rychlým tempem, přičemž většina pokroků je publikováno v odborných textech psaných v anglickém jazyce. Mnoho pojmů, které je třeba popisovat tedy ještě nemá český ekvivalent. Existují naopak případy, kdy překlad do českého jazyka způsobil koincidenci s jiným termínem (například slova *loss* i *error* jsou často v českých publikacích překládány jako *chyba*). Aby se předešlo zmatení čtenáře, budou výrazy v této práci často uvedeny v obou jazycích. V případě použití originálního anglického tvaru budou pojmy zpravidla ponechány v nesklonném tvaru.

3 Problematika malware

Slovo malware vzniklo zkrácením anglického výrazu „**Mal**icious soft**ware**“, tedy škodlivý software. Jak již původní název vypovídá, patří zde programy vykonávající jakoukoliv škodlivou nebo uživatelem nechtěnou činnost. Tou může být nevyžádané mazání dat, shromažďování citlivých informací, použití systémových prostředků k vykonávání činnosti bez vědomí vlastníka zařízení, zobrazování nechtěných reklam a mnoho dalších.

Malware lze klasifikovat pomocí více kritérií. Nejužším rozdělením je kategorizace do takzvaných **rodin** malware. Ty se tvoří na základě společných charakteristik - zpravidla vykonávají stejnou akci podobným způsobem, jsou od stejného autora nebo vycházejí z jednoho zdrojového kódu[2].

Dalším možným rozřazením je cílený operační systém. Pouze malý zlomek škodlivého kódu dokáže pracovat na více než jedné platformě. Ačkoliv v poslední době velmi rychle přibývají hrozby cílící na mobilní zařízení, nejčastěji se útočníci zaměřují na systém Microsoft Windows.[3] Tato práce bude rovněž zaměřena výhradně na tuto platformu.

Nejčastěji se spojením *typ malware* myslí klasifikace dle jejich hlavního rysu - tedy způsobu šíření, či prováděné aktivity. Je vhodné poznamenat, že následující termíny nejsou exkluzivní a moderní malware lze většinou zařadit do většího množství níže uvedených kategorií[4].

3.1 Typy malware

3.1.1 Spyware

Tento typ škodlivého programu sleduje uživatele a shromažďuje o něm prakticky jakékoliv informace bez jeho vědomí. Získaná data jsou většinou následně odesílána útočníkovi. Mezi takto zachytávané informace patří například sekvence stisknutých kláves (keylogger), spuštěné programy, navštívené webové stránky a podobné.

3.1.2 Ransomware

Jedná se o typ malware, který zpravidla zamezí použití některých zdrojů, ať už se jedná o zamknutí celého fyzického zařízení, či šifrování některých specifických souborů. Útočník pak požaduje odeslání peněžní částky na účet - výkupného (anglicky *ransom* - odtud název ransomware). K převodu dochází zpravidla v některé kryptoměně, aby se tak znesnadnilo dohledání útočníka. Až po uhrazení finančního obnosu, je opět umožněn regulérní přístup k zablokovanému zdroji. Doba, do kdy je možno svá data vykoupit, je často omezená. Při nedodržení splatnosti jsou pak informace zcela ztraceny. Tento typ malware je v dnešní době velmi populární. 12. května 2017 byl zahájen jeden z nejagresivnějších útoků vůbec. Byl způsoben programem WannaCry spadajícím právě do této kategorie.

3.1.3 Trojský kůň

Tento pojem pochází z antické báje o dobytí města Tróje. V tomto příběhu se ukryli vojáci do velkého dřevěného koně a pod záminkou daru se pak takto dostali až za brány města. Analogicky pracuje i tento typ programu, kdy si uživatel tento malware, tvářící se jako regulární bezpečný software, stáhne dobrovolně. Termín trojský kůň neříká nic o samotné nekalé aktivitě. Program se může chovat jako spyware, ransomware a tak dále.

3.1.4 Downloader

Stahovač, neboli downloader (někdy také trojan downloader) se rovněž mnohdy řadí pod trojské koně. Jeho hlavní funkcí je tajně se připojovat na vzdálený server, ze kterého tajně stahuje další malware a spouští jej.

3.1.5 Vir

Počítačový virus se do cílového zařízení dostane jako součást jiného programu. Ten se pak aktivuje spuštěním hostitele a začne se dále připojovat k dalším programům a souborům. Proto je také často místo viru používaný termín „infektor“. Ačkoliv se mnohé viry pouze připojují k hostiteli, existují i varianty, které infikovaný soubor zcela přepíší a způsobují tak škodu na datech. Tento termín bývá v českém jazyce často zaměňován s výrazem malware. V této práci se však jedná o zcela rozdílné pojmy.

3.1.6 Červ

Na rozdíl od virů se červi šíří samovolně jako samostatné soubory. Poté co dojde k infikování, začnou tyto programy využívat síťových prostředků cíle ke své další propagaci.

3.1.7 Backdoor

Česky „zadní vrátka“ slouží jako nezdokumentovaný přístup do systému napadeného, většinou otevřením příslušného portu a naslouchání na něm. Útočníkovi tak umožní vzdálené ovládání cílového zařízení bez vědomí vlastníka a většinou i bez regulérní autentizace. Velmi často je tento typ definován jako podtřída trojského koně.

3.1.8 Adware

Adware cílovému uživateli zobrazuje nevyžádané reklamy, a to buď upravováním obsahu jiných aplikací, či otevíráním vyskakovacích oken webového prohlížeče. Velmi často se do zařízení může dostat jako "volitelná"součást jiného programu díky nezkušenosti uživatele. Mnohé zdroje se rozcházejí v názoru, zda-li adware řadit mezi malware, nebo jestli jde o zvláštní kategorii.

3.1.9 Command & Control Bot

Tento software umožňuje útočníkovi použít prostředky většího množství infikovaných zařízení pro vykonání určité činnosti na povel útočníka. Tímto způsobem lze například vykonávat *Denial of Service* útoky, kdy napadený stroj začne vysílat mnoho požadavků na konkrétní server za účelem vyčerpání jeho síťových zdrojů. Nevyžádaná spojení směřované z jediného zařízení jsou velmi snadno zablokovány, avšak jelikož jsou infikovaných strojů mnohdy i tisíce, je tato obrana značně obtížnější. Další činností může být hromadné zobrazování webových stránek pro zvýšení jejich návštěvnosti, nebo použití botů k přeposílání jiného malware.

3.2 Statická analýza malware

Statická analýza provádí zkoumání podezřelého souboru bez jeho spuštění a to většinou nahlížením na interní strukturu souboru, metadata, či případný zdrojový kód.

V případě malware zaměřeného na Windows obsahují všechny tyto spustitelné soubory tzv. PE hlavičku, která je nositelem základních informací o daném souboru. Specificky zde najdeme adresy sekcí obsahující kód a data, velikosti těchto sekcí, informace o cíleném operačním systému atp. Pomocí informací v této struktuře lze nalézt tzv. tabulku importů staticky popisující importované knihovny a jejich funkce. Další informace lze získat ze struktur popisujících, jaké metody daný soubor exportuje, či jaké textové řetězce používá. V případě malware se však jejich autoři snaží bránit deklarací pouze těch nejvíce nezbytných funkcí a řetězců. Přičemž další funkce importují až v samotném běhu programu (například pomocí funkce *LoadLibraryA*).

Samotný kód programu začíná na adrese, která je dána polem *AddressOfEntryPoint*, jež se nachází v PE hlavičce. Tento kód je však v binární reprezentaci, tedy pro člověka zpravidla nečitelný. Proto se kód překládá do podoby *jazyka symbolických adres* (anglicky „assembly language“, česky se také používá výraz *assembler*). Tento proces se nazývá disassemblování (z anglického „disassembly“).

Pokud je soubor vyhodnocen jako škodlivý (ať už se jedná o důsledek statické, či jiné analýzy), bývá pro tento typ hrozby vytvořena signatura. Tou může jakákoliv struktura jednoznačně určující, že se jedná o tuto specifickou hrozbu, tudíž se obecně nevyskytuje u benigních souborů. Nejjednodušším způsobem je tedy vypočítání *hashe* celého souboru. Nevýhodou tohoto přístupu však je, že stačí přidat prázdnou operaci, pozměnit pořadí funkcí, či jakkoliv upravit soubor a dojde ke změně celého *hashe*. Alternativou je použít namísto signatury sekvenci bytů, která je typická pro analyzovaný vzorek, či skupinu vzorků. Pro větší generalizaci je pak možné ve vzorcích užít prázdné znaky, které mohou znamenat jakoukoliv hodnotu. Právě tato metoda je dnes základem mnoha antivirových programů.

3.2.1 Obrana proti analýze

Jelikož statická analýza vyplývá ze zkoumání vnitřního uspořádání souboru, snaží se tvůrci malware svá díla bránit změnou této struktury. Jednou z možností je obrana pomocí vysoké

variability kódu. Výsledkem je pak malware vyskytující se v mnoha mutacích, které mohou mít navzájem pozpřehazované funkce, různou míru „nic nedělající“ výplně, ale v komplikovanějších řešeních nemusí být evidentní jakákoliv podobnost. K tomuto účelu existuje více nástrojů - proměnlivé enkryptory, mnohonásobná permutace, generátory a další. Druhou efektivní obranou je obfuskace (také nazývaná „matení“). Jedná se o metodu, jejíž účelem nemusí být nutně obrana proti antivirovému software, nýbrž prevence reverzního inženýrství, ať už za účelem pochopení malware pro jeho lepší detekci, či napodobení chování jiným útočníkem. Pro tento účel dojde k zakódování části programu většinou užitím jedné z následujících praktik.

XOR Tato metoda používá bitovou operaci exkluzivní disjunkce (zkráceně XOR), která je postupně provedena mezi obfuskovanými bity a předem zvoleným klíčem. Na následujícím výpisu je hexadecimální reprezentace řetězce „HelloWorld“ „vyxorována“ s reprezentací sekvence čísel „123456789“. Před spuštěním takto obfuskovaného kódu je vykonáno základní makro, které nad zakódovanou oblastí provede XOR se shodným klíčem jako v případě šifrování. Až poté je celý program zahájen.

```
1 48 65 6C 6C 6F 57 6F 72 6C 64 00 00          HelloWorld..
2 30 31 32 33 34 35 36 37 38 39 00 00          0123456789..
3 78 54 5E 5F 5B 62 59 45 54 5D 00 00          xT^_[bYET]..
```

Výpis 1: Ukázka XOR obfuskace

Substituce Jedná se o běžně užívaná metodu šifrování, kdy je každý byte zaměněn za jiný dle určitého pravidla. Nejjednodušší variantou substituce je tzv. posuvná šifra, která je možná zapsat matematicky jako $(x + \text{posun})\%256$. Při spuštění programu užívajícím této jednoduché metody, nejprve dojde k „posunu“ všech hodnot o stejný počet jako v případě šifrování, akorát opačným směrem.

Base64 Base64 je formát, který se často používá k přenášení binární informace pomocí textových řetězců. Jedná se o kódování, tedy ačkoliv není reprezentace přímo čitelná člověkem, existuje konkrétní postup na převod zpět do původní podoby nevyžadující žádný tajný klíč. Ačkoliv se jedná o velmi primitivní techniku na základě známého algoritmu, i přesto je dodnes používaná. Pro stížení deobfuskace je tato metoda obvykle kombinována s již zmíněnou XOR obfuskací nebo substitucí jinou abecedou.

```
1 normal = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ+/="
2 custom = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/="
3
4 // Base64 obfuskace "HelloWorld" s "normal" abecedou -> SGVsbG9Xb3JsZA==
5 // Base64 obfuskace "HelloWorld" s "custom" abecedou -> 2q5cLqJ7LDtc9k==
```

Výpis 2: Ukázka Byte64 obfuskace se substitutcí

Komprese Mnoho dnešního malware se maskuje pomocí tzv. „packerů“, což jsou speciální algoritmy, které provedou bezeztrátovou kompresi důležitých částí daného vzorku. Díky tomu je vzorek v nespuštěné podobě velmi obtížné analyzovat. Při spuštění dojde k reverznímu procesu - dekompresi, která ve většině případů ukládá „čistou“ verzi kódu do paměti. V případě velmi rozšířených packerů dokáží komerční disassembly programy tyto algoritmy detekovat a automaticky je dekompresovat. Pokročilejší autoři však mohou použít vlastní komprimační nástroje, či upravit veřejně dostupné varianty.

3.3 Dynamická analýza malware

Dynamická neboli behaviorální analýza vychází z dat získaných zkoumáním spuštěného malwaru. Ideálně tedy dostaneme detailní informace o tom, co zkoumaný program skutečně dělal, které funkce prováděl, jaké zdroje využíval a mnoho dalších. Je však vhodné mít oddělené prostředí s cíleným operačním systémem a nástrojem, jež provede spuštění malware, jeho analýzu, uložení získaných dat do perzistentního úložiště a většinou i návrat do původního „čistého“ stavu před infekcí. Toto oddělené prostředí může být dedikovaný fyzický stroj, virtualizovaný systém, nebo dokonce pouhý program - simulace operačního systému, která nemá k dispozici všechny funkce klasické verze, avšak vyžadující minimální hardwarové zdroje. Některé zkoumané soubory mohou mít implementovanou prevenci proti dynamické analýze zkoumáním prostředí, ve kterém momentálně běží. V případě, že malware detekuje podezřelé prostředí, může přestat vykonávat škodlivou činnost a analyzátoru se tak jevit jako neškodný. Další značnou nevýhodou tohoto typu analýzy je jeho časová náročnost. Doba jednoho zkoumání může být až v řádech minut. Pokud bychom prováděli tuto akci v rámci kontroly podezřelého souboru na klientském zařízení, musíme pozdržet spouštění tohoto programu, simulovat běh v *sandboxu* a až poté informovat uživatele a pokračovat v běžném chodu. I v případě, že provádíme testování vzorků shromážděných antivirovým programem na straně vývojáře tohoto bezpečnostního prvku, pak vzhledem k faktu, že denně přibývají tisíce podezřelých souborů, bychom potřebovali mnoho strojů pracujících paralelně. Tento problém lze částečně vyřešit omezením maximální doby běhu programu na několik vteřin, případně užitím praktiky experimentující s kontrolou virtuálního času.

Další variantou, je analýza programu v reálném prostředí. Jedná se prakticky o poslední vrstvu ochrany, která může alespoň omezit škody, případně varovat další uživatele antivirových programů.

4 Problematika strojového učení

Strojové učení je podoborem umělé inteligence. Zatímco klasické algoritmy jsou sadami explicitně naprogramovaných instrukcí, strojové učení užívá statistickou analýzu k „samovolnému “ naučení problému na předem zadaných příkladech.

V závislosti na znalosti učené informace můžeme strojové učení rozdělit na dvě základní kategorie. Tou nejčastější skupinou je **učení s učitelem** (supervised learning), kdy algoritmy spadající do této skupiny přijímají za vstup dvojici - samotná data a požadovaný výsledek. Do této třídy patří například klasifikace, kdy se snažíme naučit z dat, u kterých známe patřičnou třídu, klasifikovat nové, předem neviděné data.

V druhém případě, který nazýváme **učení bez učitele**, dostávají dané algoritmy na svém vstupu pouze data bez výsledku a samy se snaží najít skryté vzory na základě kterých podobné vzorky sjednocují do skupin. Nejčastějším úkolem využívajícím toto učení je shlukování.

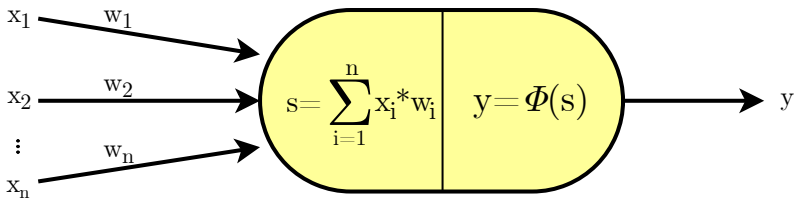
Jelikož cílem této práce je klasifikovat vybrané třídy škodlivého SW, budou dále rozebrány pouze algoritmy spadající do kategorie učení s učitelem, přičemž nejvíce prostoru v této sekci bude věnováno neuronovým sítím a zbylé algoritmy budou popsány pouze souhrnně.

4.1 Neuronové sítě

Primárním klasifikačním nástrojem použitým v praktické části této práce jsou neuronové sítě. Jedná se o výpočetní model běžně užívaný v umělé inteligenci inspirovaný biologickým nervovým systémem. Tyto sítě jsou v základu poměrně jednoduchým matematickým modelem, kde základní elementy - neurony jsou postupně spojovány ve složitější struktury. Kromě klasifikace lze tento nástroj použít pro rozpoznávání psaných znaků, objektů na fotkách, predikci trhu, kompresi obrázků a mnoho dalších.

Inspirací pro umělé neuronové sítě je organická nervová soustava. Skutečný neuron má mnoho vstupů zvaných dendrity, které přijímají elektrický vzruch a šíří jej do těla neuronu. V případě, že tento signál přesáhne určitou hodnotu, posílá jej neuron dál svým jediným výstupem - axonem, na který jsou napojeny vstupy dalších buněk.

Analogicky k nervovému systému má i každý umělý neuron několik vstupů, ačkoliv těmi jsou zde numerické hodnoty. Na všech vstupních rozhraních se vyskytují nastavitelné **váhy**, kterými je příslušná vstupní hodnota vynásobená. Tyto váhy představují učitelnou část klasifikátoru. V těle neuronu jsou všechny součiny sečteny a výsledná suma nakonec projde **aktivační** (přenosovou) **funkcí** ϕ, jejíž výsledek je finálním výstupem neuronu. Celkový postup informace jedním neuronem je znázorněn na následujícím obrázku.



Obrázek 1: Schéma umělého neuronu

4.1.1 Aktivační funkce

Možných aktivačních funkcí (někdy nazývaných *přenosové funkce*) existuje celá řada. Každá z nich má své výhody a nevýhody. Kromě podoby křivky samotné funkce je z hlediska učení důležitá i derivace těchto funkcí, která umožňuje správnou propagaci případné chyby.

Lineární funkce Nejjednodušší možná funkce daná předpisem $A = cx$. Problém této aktivace spočívá v její konstantní derivaci. Změny ve zpětné propagaci jsou tedy rovněž konstantní a zcela nezávislé na velikosti chyby. Při použití této funkce tedy nemá smysl implementace více vrstev.

Sigmoidální funkce Velmi často užívaná nelineární funkce, jejíž výstup je vždy v otevřeném intervalu $(0, 1)$. Další typickou charakteristikou je její strmý průběh při x mezi -2 a 2, což má za následek tendenci silně „přitahovat“ výsledky k jedné z jejích limit. Je tedy velmi často používaná pro binární klasifikaci. I tato funkce má však nevýhody. S rostoucí vzdáleností hodnot x od nuly dochází ke zmenšení rozdílů v hodnotách y, což způsobuje velmi malý gradient, respektive velmi malé změny v případě chyby a učení se tak značně zpomaluje.

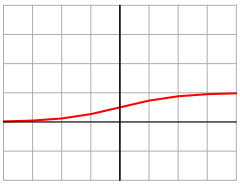
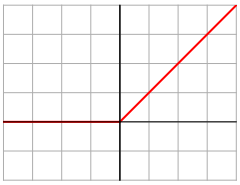
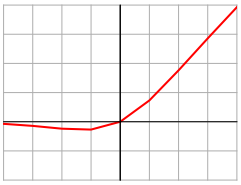
reLU Rektifikovaná lineární jednotka je v dnešní době pravděpodobně nejužívanější přenosovou funkcí. Hlavní výhodou je její jednoduchost a vlastnost utlumení záporných a nulových neuronů. Bohužel ona konstantní část reLU je zároveň nevýhodou této funkce. Problém opět tkví v derivaci, respektive v nulovém gradientu, této části. Neurony, které v průběhu učení dosáhnou nulových nebo záporných aktivací již nikdy nebudou aktivovány - dojde k jejich vymření. Pro vyřešení tohoto problému byly navrhnuty varianty této aktivace nazývané *leaky reLU,* nebo *PReLU*, které v záporné části funkce používají mírný sestup.

Swish Navržena v roce 2017 jako náhrada za reLU. Jedná se o vylepšení sigmoidální funkce, které tvarově připomíná funkci reLU. Ve srovnání s ní je však výpočetně náročnější. Autoři této aktivační funkce udávají zlepšení přesnosti v každém z jejich experimentů[5].

Softmax Jedná se o speciální funkci běžně používanou ke kategoriální klasifikaci. Jejím účelem je normalizace K-dimenzionálního vektoru tak, aby každá z jeho složek nabývala hodnot 0 až 1 a zároveň aby součet všech jeho položek se rovnal jedné. Na výstup této aktivace

lze tedy nahlížet jako na vektor, jehož každá složka vyjadřuje pravděpodobnost náležitosti do určité třídy. Tato aktivační funkce je velmi často používána jako finální funkce u výstupní vrstvy[6].

Tabulka 1: Vizualizace, rovnice a derivace aktivačních funkcí

Graf funkce	Rovnice funkce	Derivace funkce
Sigmoidální funkce 	$\phi(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$\phi'(x) = \phi(x)(1-\phi(x))$
ReLu funkce 	$\phi(x) = \max(x, 0)$	$\phi'(x) = \begin{cases} 0 \text{ pro } x \leq 0 \\ 1 \text{ pro } x > 0 \end{cases}$
Swish funkce 	$\phi(x) = x \cdot \sigma(x)$	$\phi'(x) = \phi(x) + \sigma(x)$
Softmax funkce	$\phi_i(\vec{x}) = \frac{e^{x_i}}{\sum_J^{j=1} e^{x_j}}$	$\frac{\partial \phi_i(\vec{x})}{\partial x_j} = \phi_i(\vec{x})(\delta_{ij}{}^1 - f_j(\vec{x}))$

4.1.2 Organizace neuronů do vrstev

Samotný neuron je zároveň nejjednodušším možným typem neuronové sítě, který je běžně nazýván **perceptronem**. Z jednoduché struktury tohoto modelu pak vycházejí jeho omezené klasifikační vlastnosti, jež dokáží správně rozlišit pouze lineárně separabilní problémy[7]. Pro pokročilejší případy je nutné organizovat neurony do složitější struktury, kterou obecně nazýváme „umělá neuronová síť“. Více neuronů je v tomto případě uzpůsobeno do struktury, kterou zveme **vrstva**. Každý neuron tohoto bloku dostává vstup z předchozí vrstvy a naopak předává své výstupy bezprostředně následující vrstvě. Samotná struktura a jednotlivé propojení neuronů jsou dány typem vrstvy, které jsou popsány v následujících odstavcích.

[1] Kroneckerovo delta

4.1.2.1 Vstupní vrstva

Jedná se vždy o první vrstvu sítě. Tato vrstva nemá žádné vstupní hrany, ani aktivační funkci. Počet neuronů vždy odpovídá velikosti vstupního vektoru do sítě, přičemž výstupní hodnoty jednotlivých neuronů odpovídají hodnotám příslušných složek vektoru.

4.1.2.2 Embedding vrstva

Tato vrstva je použita, pokud vyžadujeme učení sítě na diskrétních hodnotách, jejichž relace mezi sebou není zcela jasná[8]. Příkladem budiž sekvence slov v klasické větě. Jelikož síť provádí matematické operace, je nutné tato slova vyjádřit numericky. Pokud bychom použili jednoduchou mapu a jednotlivým unikátním slovům přiřadili různá čísla, splnili bychom podmínku numerických vstupů, avšak matematické operace nad těmito vstupy by nedávaly smysl.

Proto se používá odlišný princip. Každé slovo bude reprezentováno vektorem o délce *počtu unikáních slov* (v některých případech je třeba zvětšit délku o třídu reprezentující neznámé slovo), přičemž pouze jedna složka bude nastavena na číslici jedna, zbytek bude nulový. Každé unikátní slovo má přiřazenou specifickou složku ve vektoru. Tuto metodu nazýváme „kódováním 1 z n“ (anglicky *One Hot Encoding*). Výsledkem je tedy velké množství řídkých vektorů, což není příliš výpočetně efektivní. Ty jsou navíc zcela atomickými jednotkami - neexistuje žádný relevantní způsob, jak tyto vektory porovnávat.

Za účelem větší efektivity a vyjádření jednotlivých slov jako vektory, které lze mezi sebou porovnávat a reprezentovat tak jejich podobnost a asociace, vzniklo několik technik jako je například *Word2vec*[9], nebo *GloVe*[10]. Ve své podstatě obsahují tyto techniky mělké neuronové sítě, které se snaží přes jednu skrytou vrstvu naučit predikovat následující slovo v sekvenci. Ze sítě je pak extrahována ona skrytá vrstva, která slouží jako slovník pro konverzi jednotlivých slov na jejich spojitou více-dimenzionální reprezentaci. Toto vysvětlení je pouze přiblížením funkcionality těchto principů. Ve skutečnosti se jedná o neustále vylepšované sady algoritmů.

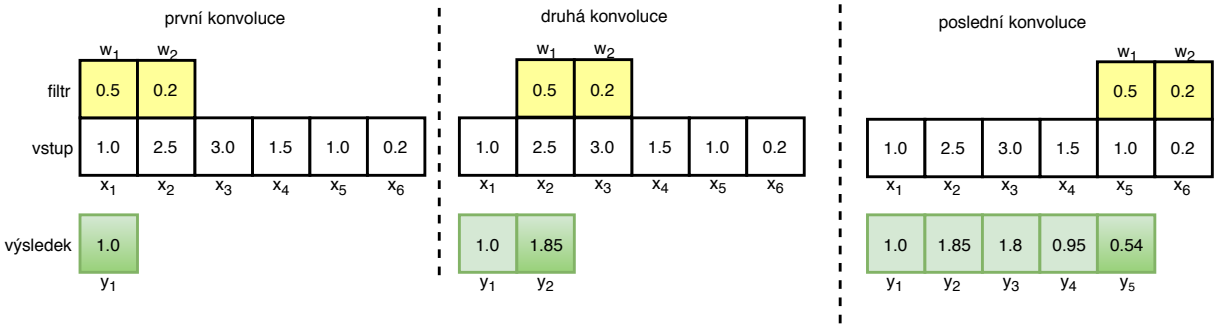
„Embedding“ v tomto kontextu znamená zmíněnou konverzi známých slov ze slovníku na spojitou reprezentaci o fixní délce. Ta může využívat přednaučeného slovníku pomocí technik *Word2vec* atp, nebo může fungovat jako klasická dopředná neuronová vrstva, jejíž váhy jsou na začátku nastaveny náhodně a postupem času se naučí jednotlivá slova vhodně konvertovat.

4.1.2.3 Plně propojená vrstva

Jak již název napovídá všechny neurony v této vrstvě přijímají veškeré výstupy z předchozí struktury. Pokud bychom vzali pouze tuto a předcházející vrstvu, pak by se z hlediska grafové teorie dalo mluvit o úplném bipartitním grafu. Ačkoliv se jedná o velmi jednoduchý princip je důležité si uvědomit, že množství hran značně rychle roste s množstvím vstupních parametrů nebo s kvantitou neuronů v této vrstvě. Velmi často se tedy stává, že právě tyto vrstvy jsou časově nejnáročnějším místem v sítí. Dalším ekvivalentním názvem tohoto uspořádání je „Dense layer“, tedy hustá, nebo hustě propojená vrstva.

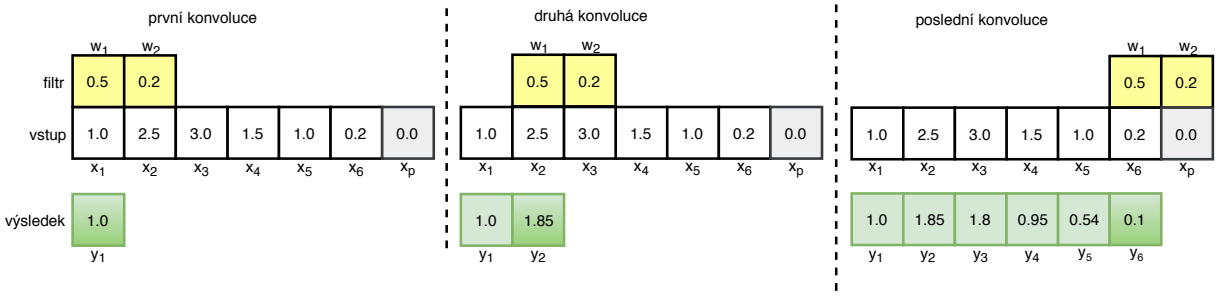
4.1.2.4 Konvoluční vrstva

Hlavním účelem této struktury je získat lokální charakteristiky vstupního objektu, respektive dílčích částí daného vstupu. Název vrstvy vychází z operací, které jsou vykonávány nad vstupním vektorem - zvané *konvoluce*. Princip tohoto úkonu spočívá v použití pohyblivého „okna“ s pevnou velikostí nazývaného **receptivní pole** (anglicky Receptive field). Zmíněné okno postupně „přejíždí“ po celé délce vstupu. Při každé iteraci této cesty se aktuální plocha stává vstupem do neuronu. Váhy jednotlivých neuronů jsou pro jednu iteraci sdílené a jejich počet je ovlivněn velikostí receptivního okna. V tomto kontextu je jedna konkrétní skupina vah označována jako **filtr** (mnohdy také zvaný jako „kernel“). Stejně jako u klasického neuronu, po vynásobení všech patřičných vstupů příslušnou vahou, dojde k celkové sumaci a průchodu aktivační funkcí.



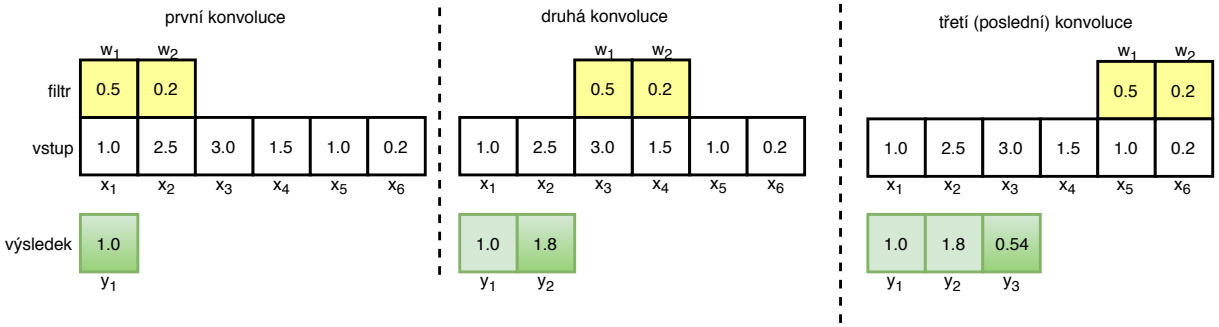
Obrázek 2: Ukázka jedno-dimenzionální konvoluce

Na předchozím obrázku lze vidět, že výsledný vektor po poslední konvoluci je o něco menší, než byl rozměr vstupu, což je ovlivněno velikostí filtru ($|y| = |x| - (|w| - 1)$). Pokud je tato redukce nechtěná, může být této vrstvě nastaven tzv. **padding** (český překlad by byl „vycpávka“). Daný parametr udává, kolik statických hodnot (zpravidla nul) má být umístěno na začátek nebo konec vstupu. Dalším důvodem proč použít tento parametr, je případná velká důležitost znaků na okraji, které by bez těchto „vycpávek“ byly použity méně-krát, než-li hodnoty více ve středu vstupu. Existuje ještě jeden důvod, proč je padding užitečný. Nejprve je ale potřebné vysvětlit význam dalšího z parametrů konvoluční vrstvy, a proto bude uveden až později. V moderních implementacích není nutné zadávat počet výplňových znaků číselně, ale stačí uvést řetězec "same", který přidá znaky střídavě na začátek i konec tak, aby měl výstup stejné rozměry jako vstup.



Obrázek 3: Ukázka jedno-dimenzionální konvoluce s výplní

Míra posunu receptivního pole je plně nastavitelná parametrem nazývaným **stride**. Většinou je hodnota této vlastnosti rovna jedné, tedy první položka v bývalém okně již nebude použita, všechny ostatní elementy dostanou váhy svého následovníka a až poslední složka vstupního vektoru je zcela nová. Nastavením tohoto parametru na stejnou velikost, jako je rozměr filtru, se zamezí překrýváním jednotlivých polí působnosti. Stride se většinou volí vzhledem k velikosti filtru. Malé filtry v řádech jednotek mívají zpravidla jednotkový posun, avšak v případě větších oken, je stride volen experimentálně. Pokud by byly zvoleny hodnoty pro posun a velikost receptivního pole „nešikovně“ může dojít ke stavu, kdy poslední vstupní hodnoty nelze využít, jelikož by posunutím okna o požadovanou míru došlo k přetečení pole. Řešením pro tuto situaci je v minulém odstavci zmíněný padding, který přidá dostatečný počet výplňových znaků na konec pole, aby nedošlo k selhání.

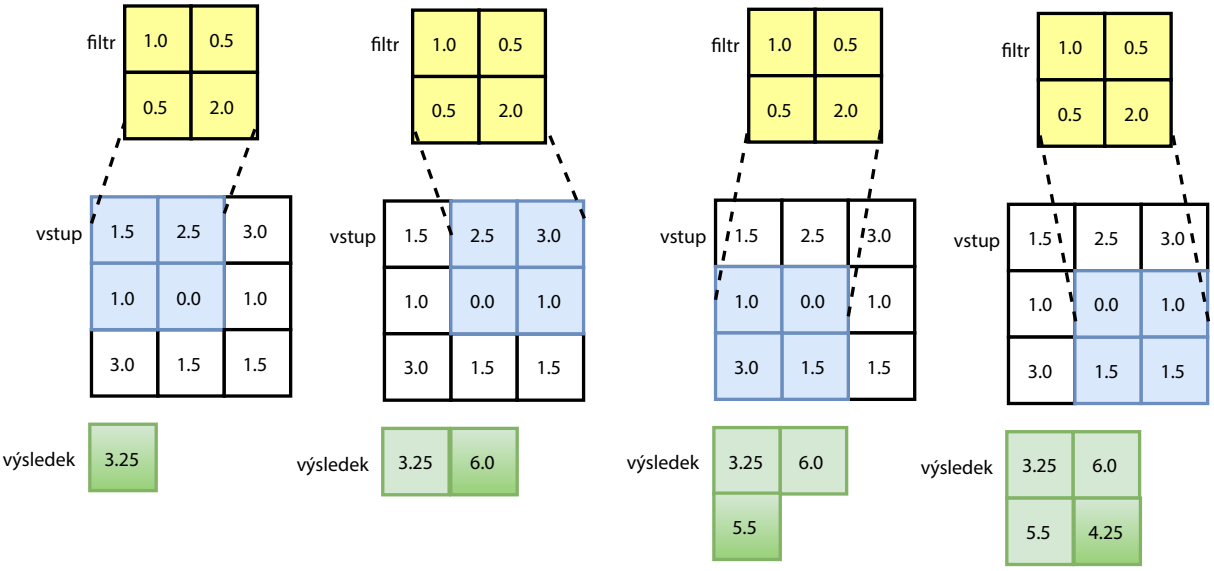


Obrázek 4: Ukázka jedno-dimenzionální konvoluce s parametrem stride = 2

Zpravidla je žádoucí mít více filtrů, tedy více sad vah. Záleží na konkrétní implementaci, zda-li dojde k vícenásobné aktivaci v jedné iteraci, nebo naopak k jedné aktivaci ve iteraci vícenásobné (většinou se užívá druhá varianta). V případě použití více než jednoho kernelu však dochází ke zvětšení dimenze výstupu. Specificky pokud vstupem byla sekvence o 200 znacích, použijeme 32 filtrů a byl zvolen „same“ padding, bude mít výstup rozměry 200x32.

Na obrázcích i v textu byla popsána konvoluce na jedno-dimenzionálním vstupu. Tuto techniku však lze použít i na více dimenzích. V umělých neuronových sítích je konvoluce nejčastěji používaná pro zpracování obrazu, tedy na dvou-dimenzionálním vstupu. Princip však zůstává

stejný. Pro jednoznačné určení velikosti filtru a stride je však nyní nutno použít dvě hodnoty (pro každou z dimenzí). V některých implementacích se připouští i jediná číslice, která způsobí vytvoření čtvercového filtru. Pohyb okna probíhá nejprve v jedné dimenzi. Jakmile je dosaženo maximálního posunu, dojde k přesunu na začátek a k pohybu v druhé dimenzi (pouze jeden krok). Nyní se postup opakuje dokud to lze.



Obrázek 5: Ukázka dvou-dimenzionální konvoluce

4.1.2.5 Pool vrstva

Tato vrstva, někdy také zvaná „subsampling vrstva“, se velmi často používá ihned po vrstvě konvoluční. Jejím účelem je zjednodušit nebo zkondenzovat informaci jdoucí ze samotné konvoluce. Myšlenkou je, že není nutné specificky vědět, kde se určitý rys nachází. Stačí pouze znát relativní pozici nebo informaci o obecné existenci (v případě velkých filtrů).

Podobně jako u konvolučních vrstev, i zde postupně posouváme pevně danou plochu a provádíme operaci na základě dat, které se vyskytují v právě zaměřené oblasti. Tato vrstva nemá žádné trénovatelné parametry. Operace je daná typem pool vrstvy.

Nejpoužívanějším typem pool vrstvy je MaxPool vrstva, která vrací maximální prvek z prozkoumávaného pole. Další častou operací je průměrování v tzv. Average Pool vrstvě a nebo L2 Pool vrstva, která vrací odmocninu sumy druhých mocnin prvků.

V případě, že vstup se skládá z více filtrovaných konvolucí, dojde k provedení operace na každé z těchto vrstev zvlášť - nedochází k poklesu dimenzí.

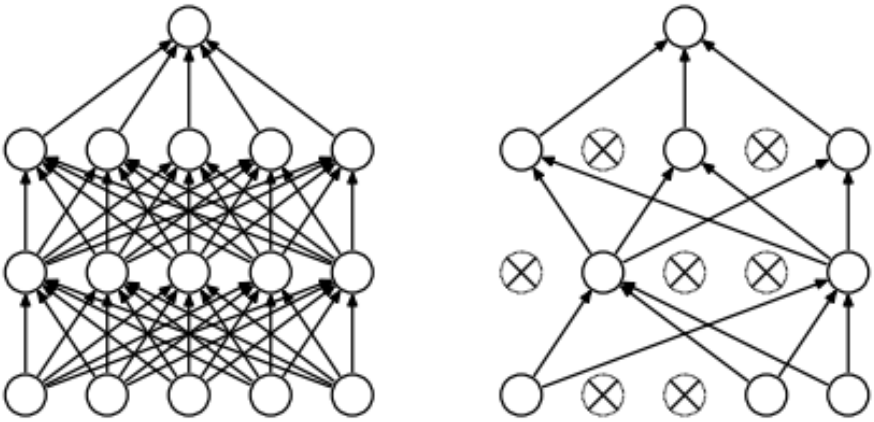
Při vybrání vhodné operace a rozměrů subsamplovacího okolí, nedojde ke ztrátě požadované informace. Naopak vlivem redukce nepotřebných informací dojde ke značnému zlepšení výkonu a větší odolnosti proti přeučení.

4.1.2.6 Flatten vrstva

Velmi primitivní vrstva, která pouze transformuje více-dimenzionální vstup na jednorozměrný vektor. Dojde tedy ke „zploštění“/„srovnání“. Jediným argument této vrstvy specifikuje pořadí, v jakém se budou dimenze redukovat.

4.1.2.7 Dropout vrstva

Dropout je princip, který byl důkladně popsán v práci autorů z torontské univerzity[11]. Má za úkol zamezovat přeučení sítě pomocí náhodné dočasné deaktivace hran s danou pravděpodobností. Pro většinu sítí je pravděpodobnost 0.5 dobrým základem. V tomto případě tedy dojde k náhodnému vybrání poloviny neuronů ve vrstvě, které budou dočasně zcela odebrány, včetně jejich vstupů a výstupů. Poté dochází ke kompenzaci - škálování zbylých neuronů o hodnotu pravděpodobnosti. Tento jednoduchý princip zabraňuje přeučení a zajišťuje tím lepší generalizaci. Dropout probíhá pouze při trénování. Evaluační proces je v tomto případě zcela nezměněn.



Obrázek 6: Ukázka principu dropoutu[11]

Dropout vrstva umožňuje použít výše zmíněný princip na hrany specifické vrstvy v síti, kdy každá tato vrstva má samostatně volitelnou pravděpodobnost vypuštění hrany. Zpravidla se používá velmi malá míra deaktivace hran po konvolučních vrstvách a vypnutí 50% neuronů z poslední nelinearity (hustě propojené vrstvy) před výstupem ze sítě.

4.1.3 Učení sítí

V případě učení s učitelem, probíhá trénování na datové sadě, u které je předem znám správný výsledek. Tato množina, nazývaná **trénovací sada**, obsahuje pro každý vstup i správný výstup. Ten je anglicky nazýván *label*, což by se do českého jazyka dalo přeložit jako „štítek“, nebo

„označení“. Vstupem je vždy vektor o konstantní délce. Výstupem je rovněž vektor, který reprezentuje korektní třídu. Pro více-třídou klasifikaci se používá takzvaný **„Kód 1 z n“** (anglicky One Hot Encoding), který mapuje dekadické číslo na binární vektor o konstantní velikosti dle celkového počtu tříd. Výsledný vektor obsahuje nuly a jedinou hodnotu 1 na pozici, odpovídající hodnotě mapovaného čísla. Tedy třída 4 z 9 možných by byla reprezentována jako „000100000“ (v závislosti na implementaci lze číslovat i „zprava“).

V předchozí kapitole bylo naznačeno, jak funguje prostup informace jedním neuronem. V případě celé sítě je tento proces opakován na všech neuronech jedné vrstvy najednou. Jakmile proběhne vypočítání všech výsledků, stávají se výstupy této vrstvy vstupy do té následující. Celý tento prostup informace bývá označován jako **dopředná aktivace**.

Na počátku učení jsou veškeré váhy sítě většinou nastavené na náhodné hodnoty, což tedy znamená, že i výsledek první dopředné aktivace bude prakticky náhodný. Každý výstup je ohodnocen pomocí tzv. „chybové funkce“ (error function). V anglické terminologii se mnohem častěji používá tzv. „loss function“, což by se dalo přeložit jako **ztrátová funkce**. Jelikož je tato práce inspirována anglickou literaturou, bude v rámci ní použit právě tento počeštěný termín. Výsledek zmíněné funkce pak bude jednoduše nazýván **ztrátou** a představuje míru odlišnosti od kýženého výsledku. Ve své podstatě je pak učení optimalizační, přesněji minimalizační problém, kdy se optimalizátor snaží najít takovou kombinaci vah, aby byla ztráta co nejnižší.

Podobně jako u aktivačních funkcí, i zde existuje více variant. Zvolení správné ztrátové funkce má podstatný vliv na výkonnost sítě. V této práci bude jako ztrátová funkce výhradně používána takzvaná křížová entropie, definovaná jako:

$$L = -\sum_{i} y_i \log p_i,$$

kde y_j představuje j-tý prvek chtěného výstupního vektoru v kódu „1 z n“ a p_j vyjadřuje pravděpodobnost náležitosti do třídy j, jež je výstupem již zmíněné aktivační funkce softmax.

Pro budoucí potřeby je vhodné zmínit derivaci této funkce:

$$L' = p_i - y_i.$$

Nejčastějším algoritmem pro minimalizaci je **metoda gradientního spádu** (Gradient descent). Ta využívá derivace předem zvolené ztrátové funkce v bodě ke zjištění rychlosti změn této funkce v daném bodě. Respektive ptáme se, jak se změní hodnota ztrátové funkce, pokud změníme konkrétní váhu o nekonečně malou hodnotu. Tedy:

$$\frac{\partial L_{j_n}}{\partial w_{i_m \to j_n}}$$

Použitím řetízkového pravidla:

$$\frac{\partial L_{j_n}}{\partial w_{i_m \to j_n}} = \frac{\partial L_{j_n}}{\partial y_{j_n}} * \frac{\partial y_{j_n}}{\partial s_{j_n}} * \frac{\partial s_{j_n}}{\partial w_{i_m \to j_n}}$$

Jedná se tedy o součin parciální derivace ztráty neuronu v závislosti na jeho aktivační hodnotě y, parciální derivace aktivační funkce y v závislosti na sumě vstupů s a konečně parciální derivace sumy vstupů s v závislosti na konkrétní váze w. Pokud budeme postupovat pozpátku, pak ztráta neuronu v závislosti na jeho aktivační hodnotě je pouhou derivací ztrátové funkce.

$$\frac{\partial L_{j_n}}{\partial y_{j_n}} = L'(y_{j_n})$$

Parciální derivace aktivační hodnoty neuronu v závislosti na sumě jeho vážených vstupů s_{j_n} lze vyjádřit pomocí derivace aktivační funkce.

$$\frac{\partial y_{j_n}}{\partial s_{j_n}} = \phi'(s_{j_n})$$

Nakonec parciální derivace sumy vstupů neuronu v závislosti na jedné konkrétní váze z neuronu i_m do j_n můžeme zapsat jako aktivační hodnotu předešlého neuronu y_{i_m}.

$$\frac{\partial s_{j_n}}{\partial w_{i_m \to j_n}} = y_{i_m}$$

Pomocí tohoto postupu lze vypočítat chybu neuronů ve výstupní vrstvě. Všechny zderivované funkce jsou k dispozici v předchozích odstavcích této kapitoly.

K samotné úpravě vah pomocí této chyby lze využít více vztahů. Ten nejzákladnější a nejčastěji užívaný je zobrazený níže. Kromě předem zmíněných hodnot se zde uplatňuje koeficient η, který zveme **koeficientem učení**. Ten vyjadřuje, k nakolik velké změně dojde. Pokud by byl zvolen velmi vysoko, bude učení značně nestabilní.

$$W_i^{t+1} = W_i^t - \eta \frac{\partial L(W_i^t)}{\partial W_i}$$

Pro všechny nižší (skryté) vrstvy probíhá tento výpočet obdobně s tím rozdílem, že výstup této vrstvy ovlivňuje neurony všech vrstev následujících, a tedy chybou l je nutné brát sumu všech chyb neuronů zmiňované následující vrstvy. To je také důvodem, proč se počítá chyba od výstupních vrstev směrem k vstupním. Všechny potřebné hodnoty jsou již spočítány v předchozím kroku. Z tohoto principu také plyne název procesu - **zpětná propagace** (*Backpropagation*).

Stejně jako v případě výpočtu gradientu výstupní vrstvy platí následující vzorec.

$$\frac{\partial L_k}{\partial w_{i_m \to j_n}} = \frac{\partial L_k}{\partial y_{j_n}} * \frac{\partial y_{j_n}}{\partial s_{j_n}} * \frac{\partial s_{j_n}}{\partial w_{i_m \to j_n}}$$

Avšak parciální derivace ztráty vzhledem k aktivaci, je nyní nutno vyjádřit, jako sumu ztrát všech neuronů vrstvy následující.

$$\frac{\partial L_k}{\partial y_{j_n}} = \sum_{i=0}^{|k|} \frac{\partial L_{k_i}}{\partial y_{k_i}} * \frac{\partial y_{k_i}}{\partial s_{k_i}} * \frac{\partial s_{k_i}}{\partial w_{j_n \to k_i}}$$

V závislosti na frekvenci provádění optimalizace vah můžeme identifikovat tři typy gradientního spádu.

Batch gradient descent Klasická forma gradientního spádu. Optimalizace je provedena vždy na konci epochy. Ke zpětné propagaci chyby dochází pro každý vzorek. Výsledné hodnoty jsou na konci každé epochy zprůměrovány. Důsledkem jsou pouze malé změny, což může způsobit značně pomalý průběh trénování sítě. Další nevýhodou je velká paměťová náročnost. Ukládat všechny změny pro velkou síť a rozměrný dataset, může být prakticky nemožné. Dalším nedostatkem je nemožnost přidávat nové vzorky v průběhu učení.

Stochastic gradient descent Jedná se o opak algoritmu batch gradient descent. Po každé dopředné aktivaci nastává výpočet delty vah a následně pak k aktualizaci všech vah pomocí této hodnoty. Následkem je pak nezanedbatelné kolísání celkové ztráty napříč učením. To však umožňuje síti skočit do potencionálně lepší minimální konfigurace.

Mini-batch gradient descent Poslední varianta kombinuje výhody obou předchozích případů. Kompletní dataset je rozdělen na více podmnožin, kdy má až na výjimky (při použití rozměru, který nedělí kompletní velikost beze zbytku, je poslední množina menší) každá z nich stejnou velikost. K aktualizaci vah dochází na konci trénování sítě na jedné podmnožině. Většina moderních nástrojů pro modelování neuronových sítí používá tento princip automaticky s využitím optimalizovaných množinových operací. Proměnná, která ovlivňuje velikost většiny podmnožin, se označuje jako **batch size**.

Ve většině moderních frameworků se používá varianta mini-batch gradient descent, přičemž samotný algoritmus bývá označován jako SGD, tedy stochastic gradient descent, jehož parametr „batch size“ udává velikost dávky. Tento parametr může být roven jedné, čímž vynutíme změnu váhy po každém vzorku.

4.1.3.1 Gradientní optimalizátory

Kromě klasického algoritmu pro výpočet delty vah existují další specializovanější algoritmy, které mohou učení značně urychlit, nebo dosahují lepšího optima oproti klasickému gradientnímu sestupu[12]. Stejně jako v případě předchozích zmíněných variant nebudou tyto názvy překládány.

Gradient descent with momentum Klasická varianta gradientního sestupu má tendenci oscilovat v místech, kdy jedna dimenze klesá mnohem rychleji než ostatní. Pro tyto případy, ve kterých se algoritmus nachází v okolí připomínající údolí nebo žlab, bylo navrženo jednoduché vylepšení užitím tzv. „hybnosti“ (*momentum*). K tradičně vypočítané změně

váhy se ještě přičte část μ změny z předchozí iterace. Analogií pro tento případ může být kutálení míče z kopce. V tomto případě míč postupně zrychluje, dokud nedosáhne terminální rychlosti. I po skončení „kopce“ a nástupu roviny si míč uchovává hybnost a může se tak dostat do nových minim i přesto, že aktuální gradient je nulový.

$$\delta_{t+1} = \mu\delta_{t-1} + \gamma\Delta_w L(W_t)$$

Nesterov accelerated gradient Tato varianta navazuje na předchozí vylepšení pomocí hybnosti. Dochází zde k poměrně malé změně ve výpočtu gradientu. Namísto počítání gradientu současné pozice, je zde kalkulován gradient místa, ve které bychom se ocitli použitím klasického algoritmu *gradient descent with momentum*.

$$\delta_{t+1} = \mu\delta_{t-1} + \gamma\Delta_w L(W_t - \mu\delta_{t-1})$$

Adagrad Namísto zakomponování setrvačnosti, se zde pro vylepšení rychlosti optimalizace používá proměnlivý koeficient učení. Pro každou váhu v síti je uchovávána suma čtverců bývalých gradientů s. Dynamický koeficient je pak vypočítán následujícím vzorcem:

$$\mu' = \frac{\mu}{\sqrt{s + \epsilon}}.$$

Hodnota ϵ v předchozím vzorci značí arbitrární mikroskopickou hodnotu zamezující dělení nulou. Z tohoto vzorce rovněž vyplývá možná nevýhoda algoritmu adagrad. Jelikož se penalizační suma v průběhu učení pouze navyšuje, může v extrémním případě nastat úplné potlačení jakýchkoliv změn v síti.

RMSprop RMSprop vznikl jako pokus o negaci výše zmíněné vady v algoritmu Adagrad. V tomto případě se pro výpočet dělitele koeficientu učení nebere celá suma všech čtverců, ale pouze jeho zlomek, přičemž zbývající díl je dopočítán z aktuální sumy - používá se tedy takzvaný klouzavý průměr.

Adam Poslední zmíněný algoritmus nese název *Adaptive Moment Estimation* - zkráceně Adam. Jedná se o variantu kombinující metodu *gradient descent with momentum* a *RMSprop*. Existuje i případ používající Nesterovu akceleraci pojmenovaný **NAdam**.

4.1.3.2 Negradientní optimalizátory

Jak již z názvu vyplývá, do této kategorie spadají algoritmy, jež k nalezení optimálního řešení problému nepoužívají informace z derivací. Možných metod je velmi mnoho a většina z nich se přestala používat při „objevení“ techniky zpětné propagace. Jejich nevýhoda spočívá ve špatném škálování. Jinými slovy schopnost těchto algoritmů najít vhodná řešení, značně klesá s rostoucí velikostí problému[13]. Vzhledem k tomu, že sítě mohou obsahovat až řádově miliony optimali-

zovatelných vah a samotná dopředná aktivace sítě na všech datech může trvat až řádově desítky minut, nejsou negradientní optimalizátory vhodné pro účely této práce.

Funkcí spadajících do této kategorie je velmi mnoho. Obecně zde můžeme zařadit stochastické optimalizační algoritmy, jako je simulované žíhání, random search, hejnové algoritmy nebo například evoluční algoritmy.

4.1.3.3 Pomocné prvky a techniky

Kromě výše uvedených algoritmů s adaptivním koeficientem učení, můžeme podobného výsledku dosáhnout i se standardním gradientním spádem pomocí řady dalších technik.

Decay / Úpadek Tímto pojmem se rozumí snižování koeficientu učení pomocí funkce, která má za argumenty aktuální hodnotu zmíněného koeficientu a čas (pořadí momentální epochy). Dle použité funkce lze rozlišovat několik typů úpadku. Pokles koeficientu může být lineární, po krocích (co k epoch dojde ke snížení míry učení na určitý zlomek původní hodnoty), nebo například exponenciální.

Plánovač koeficientu učení Jedná se o metodu, jež je podobná úbytku po krocích. Rozdílem je pevné stanovení specifických epoch, kdy ke změnám dojde a konstantní změna. Může tedy dojít ke zvýšení i snížení koeficientu učení.

Redukce koeficientu učení při stagnaci Jak již název napovídá, tato metoda sleduje vývoj měřené hodnoty (přesnost, nebo ztráta testovací sady) a v případě stagnace (nezlepšení) po stanovenou dobu (počet epoch) dojde k redukci koeficientu o pevně daný poměr.

Nepřímo pomoci s učením mohou i další praktiky.

Předčasné ukončení učení Podobně jako redukce LR při stagnaci, i zde je pozorována konkrétní hodnota a pokud nedojde po určitou dobu ke zlepšení výsledku, učení se automaticky zastaví.

Checkpoint / kontrolní body Při učení jsou v paměti ukládány váhy, s jejichž použitím došlo k dosažení nejlepšího výsledku. V případě, že dojde k zastavení učení v neoptimálním stavu, dojde k načtení této nejlepší konfigurace.

Přeuspořádání trénovací množiny V závislosti na užité frekvenci změny vah může pořadí vzorků v trénovací množině ovlivňovat rychlost učení. Ve většině případů se ideálních výsledků docílí náhodným přeuspořádáním pořadí jednotlivých prvků při trénování v každé epoše. Největší význam má náhodné pořadí za použití aktualizace vah po jednotlivých mini-dávkách. Mohlo by se stát, že celá tato podmnožina obsahuje vysoce korelující data, která se běžně v obecné testovací množině nevyskytují a nedojde ke kýžené generalizaci. Experiment provedený v roce 2009[14] odzkoušel zcela odlišný přístup k pořadí učení. Metoda pojmenovaná „Curriculum Learning“ (v českém jazyce by překlad mohl znít „Učení dle osnovy“) byla inspirována učením žáků ve školách, kdy se při osvojování nových technik učí

nejprve nejjednodušší příklady, na které pak navazují úkoly pokročilejší. V případě učení neuronových sítí rozpoznávat geometrické obrazce bylo nejprve vyzkoušeno standardní učení na celkové datové sadě. Druhý pokus pak zahrnoval rozdělení trénovací množiny na dvě části, přičemž ta první obsahovala pouze základní obrazce a ta následná zahrnovala i deformované tvary a atypické příklady. Učení pomocí této „osnovy“ dosahovalo lepších výsledků.

Normalizace dávky Tato technika byla poprvé představena v práci Google zaměstnanců Ioffe a Szegedy v roce 2015[15]. Podobně jako dochází k normalizaci vstupů do neuronové sítě, je v zde navrhováno normalizovat i vstupy do všech dalších vrstev neuronové sítě. Použití normalizace snižuje efekt, který autoři pojmenovali „internal covariant shift“. Jeho redukcí dochází k značnému urychlení konvergence sítě a k regularizaci na podobné úrovni, jako by zajišťovala Dropout vrstva. Ve frameworku Keras je tato metoda zakomponovaná do vrstvy o stejném názvu.

4.1.3.4 Přeučení

Přeučení nastává v případě, kdy se neuronová síť začne učit příliš detailní rysy trénovací sady, což způsobí ztrátu schopnosti generalizovat. Analogií by mohlo být učení studentů „nazpaměť“, místo učení se podstatě problému. Takto vzdělaný žák dokáže správně vyhodnotit již vyskytnuté problémy, avšak nové variace příkladů vyřešit nedokáže. Nejčastěji k tomuto problému dochází, pokud je kvantita trénovatelných parametrů větší, než-li počet možných relevantních informací v trénovací množině. Jinými slovy síť je mnohem komplikovanější, než stačí pro natrénování zvoleného problému.

Nejčastější řešení tohoto procesu je popsáno v bodech níže.

Změna struktury Protože zbytečná komplexita sítě je nejčastějším důvodem přeučení, pak eliminace této příčiny je nejočividnějším řešením problému. Bohužel ne vždy je tato cesta jednoduchá. Příliš velkým zjednodušením můžeme způsobit nedostatek informací ke zjištění globálního optima.

Dropout Tento způsob byl přiblížen v odstavci 4.1.2.7 a je velmi efektivním způsobem, jak zabránít přeučení.

Předčasné ukončení učení Tato metoda zmíněná již v předchozí sekci o pomocných technikách sleduje přesnost či ztrátu na testovací množině. Pokud začne docházet ke zhoršení dané veličiny, bude síť zastavena předčasně. V kombinaci s pamatováním si nejlepší historické konfigurace pak může nastat záměna množiny vah a návratu do předchozího optimálního stavu.

L1 a L2 regularizace Tyto dvě metody upravují ztrátovou funkci přidáním tzv. regularizační komponenty, která penalizuje komplikovaná řešení nebo příliš velké váhy. Následující rovnice popisují zmíněné L1 (vzorec 1) a L2 (vzorec 2) komponenty.

$$L(W)^* = L(W) + \lambda \sum_{j=1}^{n} |W_j| \,, \qquad (1) \qquad\qquad L(W)^* = L(W) + \lambda \sum_{j=1}^{n} W_j^2 \,. \qquad (2)$$

4.1.4 Organizace vrstev do sítě

Popsané vrstvy nejsou do sítě uspořádány náhodně. Během vývoje oboru strojového učení došlo k identifikaci specifických konfigurací, které dosahují dobrých výsledků. Ty pak byly generalizovány a ve variacích se objevují ve více experimentech. Cílem tohoto odstavce je jednoduše popsat vybrané generalizace architektur, které jsou využitelné pro klasifikaci malware.

Nejjednodušší architekturou co se popisu týče jsou sítě skládající se výhradně z dopředných hustých vrstev. V této struktuře je tedy každý neuron jedné vrstvy spojen s každým neuronem vrstvy následující. S rostoucí šířkou (a do menší míry i hloubkou) sítě pak značně stoupá i počet trénovatelných vah.

Sítě, jež používají konvoluční vrstvy k učení rysů s postupně narůstající komplexitou, nazýváme *Konvoluční neuronové sítě* (Convolutional Neural Network - CNN). Jejich obecná struktura vychází z práce, která byla publikována v roce 1998 s názvem „Gradient-Based Learning Applied to Document Recognition“[16]. V oné době byly výpočetní zdroje značně omezené, proto byla snaha této práce redukovat množství výpočetních operací nutných ke zpracování strukturovaných dvou-dimenzionálních dat. Navrhnutá síť používala opakování konvolučních a pooling vrstev k extrakci rysů z obrazu a poté až hustých vrstev pro samotnou klasifikaci.

Zmíněná architektura je pořád stále používaná a stala se téměř synonymem k výrazu *konvoluční síť*. V moderní době došlo k vylepšení modelu pomocí dropout vrstev s malou pravděpodobností zahození informace ihned po konvolucích a Dropouty s velkou pravdpodobností po klasifikačních hustých vrstvách.

V nedávné době se složitost těchto sítí rozšířila o nelineární filtry, větvení a zanořování. Tyto typy sítí jsou zaměřeny na klasifikaci a identifikaci objektů v obrázcích a jsou pro účely rozpoznávání malware zbytečně komplikované.

Práce „Language Modeling with Gated Convolutional Networks“[17] se pokusila zkombinovat výhody rekurentních a konvolučních vrstev. Výsledná architektura byla nazvána „Gated Convolutional Neural Network“ (zkráceně **GCNN**). Její princip spočíval v použití dvou paralelních konvolučních vrstev, kdy jedna z nich používala sigmoidální aktivační funkci. Poté byly jednotlivé výstupy z obou vrstev mezi sebou vynásobeny. Ve výsledku tedy sigmoidální aktivace (výstupy nabývají hodnot 0 až 1) sloužila k omezování významnosti některých informací plynoucí z druhé větve. Omezovací část je zde označována jako „výstupní hradlo“ (anglicky *Output gate*), odtud vyplývá název této architektury.

4.1.5 Hledání optimálních parametrů

I přes nedávný vývoj v oboru umělých neuronových sítí neexistuje pravidlo, které by určilo, jak by měl vypadat ideální model pro specifický úkol a jaké by měly být jeho parametry. Exis-

tuje více způsobů, jak optimalizovat navrženou konfiguraci. Učení sítě můžeme ovlivnit změnou počtu vrstev, jejich parametrů (počet neuronů, velikost filtrů atp.), učících funkcí (algoritmus, koeficient učení).

4.1.5.1 Random Search

Nejprimitivnějším algoritmem pro vyhledání optimálního nastavení, je tzv. *náhodné prohledávání* („Random search“). Jak sám název napovídá, tato metoda náhodně volí parametry, provede s jejich použitím experiment a následně vyhodnotí úspěšnost. Pokud je výkon sítě lepší než předchozí nejlepší varianta, bude tento výsledek a jeho zdrojové parametry zapamatovány.

4.1.5.2 Grid Search

Nejpoužívanější metoda optimalizace je označována jako „Grid search“ a spočívá v prohledávání všech kombinací zadaných variant parametrů. Pro plné využití tohoto algoritmu je nutné mít představu o použitelných parametrech. Pro 3 zvolené různé koeficienty učení a 3 různé počty neuronů, by tento algoritmus vygeneroval 9 variantních konfigurací. Na všech z nich by bylo vykonáno učení a každý pokus byl ohodnocen uživatelem stanovenou metrikou, která zpravidla zahrnuje poměr mezi výslednou přesností a časem učení.

4.1.5.3 Genetický algoritmus

Další možností je použít algoritmus z rodiny evolučních algoritmů. Jeho princip je inspirován přírodním výběrem. Na začátku je zvolen daný počet jedinců - kombinací parametrů, které byly získány manuálně, či náhodně. Celý tento soubor nazýváme populací a jedna konkrétní iterace populace je pojmenována generací. Sítě se naučí použitím parametrů těchto jedinců a každého z nich vyhodnotí pomocí předem stanovené metody (tzv. *Fitness function*) - tou může být dosažená přesnost, ztráta, či poměr přesnosti a času učení atp. Nyní dochází k výběru nejlepších jedinců dle tohoto zvoleného skóre - zvolíme fixní procento z nich a zbytek generace zahodíme. Nová populace je vytvářena náhodnou kombinací parametrů ze zvolených přeživších jedinců procesem zvaným *křížení*. Tímto způsobem však pouze dochází k náhodnému grid search. Abychom získali nové, unikátní, doposud neprozkoumané kombinace, dohází ještě k procesu zvanému *mutace*. Novému jedinci se s určitou (zpravidla malou) pravděpodobností změní zděděná vlastnost. Způsobů této změny je řada u reálných hodnot však většinou dojde k přičtení, či odečtení náhodné hodnoty ve zvoleném rozsahu.

Existují i další algoritmy z rodiny evolučních algoritmů jako je genetické žíhání, diferenciální evoluce a mnoho dalších.

4.2 Support Vector Machine

V základní podobě se tento algoritmus snaží nalézt nadrovinu, která by prostor zvolených příznaků rozdělila na dvě poloviny reprezentující konkrétní třídy. K definici nadroviny je zapotřebí

brát v potaz pouze prvky ležící nejblíže k ní. Množina těchto nejbližších bodů tvoří tzv. podpůrné vektory, což dalo název této metodě. Dělící nadrovina je volena na základě dvou faktorů - tak, aby byl každý vzorek správně klasifikován a zároveň tak, aby vzdálenost vzorků obou tříd byla od dělícího prvku co největší. V praxi je tento problém minimalizací kriteriální funkce (cost function), která vyjadřuje kompromis mezi těmito faktory.

Ve složitějším případě nemusí být dělícím prvkem lineární geometrický objekt, ale mohou být použity polynomiální děliče, či rozdělení na radiální bázi. Druhým řešením nelineárního problému je přetransformovat stávající data na vektorový prostor vyšší dimenze, ve kterém jsou zachovány lineární relace a separovat chtěné třídy v takto vytvořeném novém prostoru. Zde však nastává problém s výpočetní náročností vysoko-dimenzionálních dat, který je vyřešen použitím **jádrové funkce** (kernel function), tedy funkce, která bere za vstup vektory v originálním prostoru a vrací skalární součin v prostoru zvolených příznaků. Následkem toho tento klasifikátor nepracuje ve vysoko-dimenzionálním prostoru, ale stačí pouze výpočet skalárního součinu mezi vektory onoho vektorového prostoru.

Rozšířením algoritmu můžeme aplikovat princip SVM i na klasifikaci více tříd[18][19]. Dva běžně užívané přístupy jsou nazývány One-Against-One (zkráceně 1A1, česky jeden proti jednomu) a One-Against-All (zkráceně 1AA, česky jeden proti všem). Technika 1AA je historicky nejstarší a spočívá v rozdělení N-třídní klasifikace do N binárních klasifikací (patří do třídy „k“) nebo patří do „jakékoliv kromě k“). Modernější přístup, který je často používaný v dnešních implementacích, spočívá v sestavení všech možných dvojic klasifikátorů - tedy $k(k-1)/2$. Konkrétní třída je přiřazena hlasováním všemi klasifikátory, kdy vyhrává skupina s nejvíce hlasy. Z počtu klasifikátorů vyplývá, že případ 1A1 je výpočetně jednodušší, jeho nevýhodou je však pokles výkonu při užití datasetů, jež nemají homogenní počet vzorků ve třídách.

4.3 Rozhodovací strom

Rozhodovací strom je poměrně jednoduchý rekurzivní algoritmus, který postupně rozděluje množinu dat na menší podmnožiny do té doby, než zbudou podmnožiny obsahující jednu jedinou třídu. Každý uzel vytvářeného stromu představuje konkrétní příznak a pomocí stanovené hranice dojde k rozdělení množiny na dvě podmnožiny, které buď obsahují nižší, nebo vyšší hodnoty než-li je hraniční hodnota. Výběr každého uzlu se provádí užitím statistické metody, kdy je cílem dosáhnout takového rozdělení vytvářejícího co „nejčistší“ podmnožinu - ideálně obsahující pouze prvky jediné třídy. Konkrétních algoritmů pro volbu takového uzlu existuje velká řada. Ty nejčastější užívají entropii, chi-kvadrát, nebo Giniho index. Při komplikovaných klasifikacích mají stromy tendenci narůstat do komplexních tvarů, což má za následek přeučení. Byly proto navrhnuty algoritmy, které stromy zpětně pročišťují od sekcí, které mají velmi malou klasifikační schopnost.

4.4 Náhodný les

Jak již název vypovídá, tento algoritmus navazuje na předchozí rozhodovací stromy. Namísto tvoření jediného komplexního stromu pokrývající všechny možné atributy a třídy, se konstruují jednoduché stromy, které volí pouze z omezeného, náhodně zvoleného, počtu příznaků. Po sestavení všech stromů dojde k průchodu dat všemi těmito grafy. Každý z nich hlasuje pro určitou třídu, přičemž ta s většinou hlasů vyhrává. Náhodný les je velmi rychlým algoritmem, který je díky velkému počtu subklasifikátorů značně odolný vůči přeučení.

4.5 Boosting algoritmy

Rodina boosting algoritmů je poměrně novou a hodně zkoumanou skupinou klasifikátorů. Podobně jako náhodné lesy tyto algoritmy sestavují více „méně kvalitních“ klasifikátorů, které pak spolupracují k dosažení lepších výsledků. Zatímco se však náhodné lesy skládají z mnoha samostatných celků, v případě boosting algoritmů jsou tyto části sekvenční - jeden klasifikátor se učí z chyb předešlého. Nejčastěji používanými klasifikátory v této skupině je Gradient Boosting, Adaboost a XGBoost.

5 Současné publikace

V této sekci budu popisovat momentální state of art, tedy stávající odborné dokumenty zabývající se podobnou tématikou jako tato diplomová práce.

5.1 Klasifikace malware jako obrazu

Autoři této práce[20] se zaměřili na vizualizaci malware a následnou automatickou klasifikaci za pomocí těchto dat. Podezřelý program je sekvenčně čten a hodnota každého 1 byte odpovídá intenzitě černobílého pixelu ve výsledném obrazu. Spousta tvůrců malware snaží bránit detekci pozměněním malých částí kódu případně pozpřehazováním instrukcí. Ačkoliv se změny ve výsledném obrázku projeví, stále by měla být znatelná globální struktura programu a jednotlivé vizualizace jedné rodiny malware by měly být dostatečně podobné, aby je mohl automatický klasifikátor správně rozřadit. 2D struktura obrazu se vytvoří specifikováním pevné šířky dle celkové velikosti malware, výška je vždy variabilní. Mnoho klasifikátorů potřebuje vstupy stejných rozměrů, aby mohly správně určit podobnost. V této práci autoři použili algoritmus GIST k extrakci 320 příznaků, které pak do tříd klasifikovali pomocí algoritmu k-nejbližších sousedů. Další obranou proti detekci automatizovanými systémy je obfuskace použitím packerů. Rodiny používající jeden konkrétní typ packeru jsou si však stále vizuálně podobné s originálem. Pokud je zvolena velká míra komprese, pak jsou obfuskované vzorky podobné mezi sebou.

5.2 Klasifikace malware pozřením celého EXE

Poměrně nová studie[23] publikovaná na vývojářském blogu výrobce grafických karet nvidia se zaměřuje na statickou analýzu malware pomocí neuronových sítí bez jakéhokoliv předzpracování či dokonce znalostí funkce malware a extrakce jejich příznaků. Vstupem do sítě je v tomto případě kompletní binární reprezentace zkoumaného spustitelného souboru. Daný vstup je dále tokenizován po jednotlivých bytech a následně „embeddován“(viz paragraf 4.1.2.2). Následuje konvoluce s bránou (anglicky gated convolution), kdy je tok duplikován a vstupuje do dvou odlišných jedno-dimenzionálních konvolučních vrstev, přičemž jedna z nich má aktivační funkci reLU a druhá je aktivována sigmoidální funkcí. Výstupy obou těchto konvolucí jsou mezi sebou vynásobeny, projdou další reLU aktivací, globálním maximálním poolingem a přes hustě propojenou vrstvu končí ve dvou-dimenzionální vrstvě se softmax aktivací. Na gigantickém počtu dvou miliónů vzorků se síť učila celý měsíc a to na 8 velmi výkonných kartách určených přímo pro deep learning (NVIDIA Tesla M40). Výsledkem však byla síť dosahující přesnosti přes 90%, jež je velmi odolná proti overfittingu, respektive vykazuje velmi dobré generalizující vlastnosti. Součástí práce je i velmi podrobné srovnání s dalšími moderními architekturami neuronových sítí a manuální analýza této sítě popisující které sekce mají vliv na výslednou klasifikaci.

5.3 Malware a fraktály

Další nová práce, která byla publikována v únoru roku 2018[24] se přímo netýká neuronových sítí, avšak stále pojednává o problému klasifikace malware. Specificky se v této práci autoři zaměřují na charakteristiku rozhodovací množiny mezi benigním software a malware. Jejich hypotézou je, že tato množina má fraktální charakter. Tu dokazovali experimentem na poměrně malém datasetu, avšak se vzorky z velikého rozsahu co se týče stáří. Datová sada tak obsahovala například vzorky malware červa „Creeper“ z roku 1971, viru „ILOVEYOU“ z roku 2000, ale i Trojana „Zeus“, který vznikl v roce 2007, ale je stále velmi aktivní. Původní množství 551 souborů bylo zredukováno na pouhých 180, vlivem vynechání kódu, který nebyl založený na „PE32“, nebo vyčleněním kódu, který byl komprimován, či obfuskován. Z výsledných spustitelných souborů byly vyextrahovány sekvence operací v jazyce symbolických adres, které byly přes „Greyův kód“ převedeny na reálná čísla. Z nich byla vypočítána tzv. Heptor statistika, která byla použita jako základ pro vypracování Bayes a Data Model klasifikátorů. Postupným zlepšováním modelu došli autoři k závěru, že povaha hranice pomezí rozhodovací množiny mezi jednotlivými malware rodinami a obecnou skupinou reprezentující benigní vzorky, je fraktálního charakteru. Dále byla odvozena rovnice, která mohla klasifikovat jednotlivé vzorky do patřičných rodin se 100% úspěšností.

5.4 Použití grafové teorie ke klasifikaci malware

Práce pocházející z indického ústavu Amrita Vishwa Vidyapeetham[21] pohlíží na detekci malware jako na problém teorie grafů. Zkoumané soubory jsou nejprve disassemblovány. Získané sekvence operací jsou převedené do formy grafu, kde uzly představují lineární sekvenci operací a hrany reprezentují přeskoky mezi nimi pomocí instrukcí *jmp* (nepodmíněný skok), *jcc* (podmíněný skok), *call* (volání funkce) a *return* (návrat z funkce). Výsledný graf je dále pročištěn od nedosažitelných struktur, mnohonásobných skoků, atp. V rámci klasifikace se pak na problém nahlíží jako na hledání izomorfních podgrafů, které reprezentují například dešifrovací procedury.

Podobných prací na toto téma je větší množství. Jejich motivací je menší náchylnost k falešným poplachům. Ačkoliv se v základu nejedná o problematiku strojového učení, nedávná práce[22] prezentována v rámci mezinárodní konference ICISSP 2017 aplikovala tento postup s využitím SVM klasifikátoru na téměř 5 000 vzorků, ve kterých byli autoři schopni rozpoznat malware v 99% případů. Stejný princip byl aplikován pro správnou klasifikaci do 13 typů dle chování malware, kdy bylo dosaženo 96% přesnosti a finálně byly vyzkoušeny generátory virů, které tento klasifikátor byl schopen odhalit v úctyhodných 100% případů. Pro srovnání komerční software společnosti McAfee měl úspěšnost 96% a antivirus české firmy Avast dosáhl 87% přesnosti.

6 Experimentální část

6.1 Platforma

Platformou v tomto kontextu je myšlen soubor výpočetního zařízení - tedy hardware a užitých knihoven a nástrojů - software. Veškeré výpočty probíhaly na osobním zařízení, které nebylo sestaveno za účelem takto náročných výpočtů.

Tabulka 2: Specifikace výpočetního zařízení

Základní deska	GIGABYTE Z370 Aorus K3
Procesor	Intel Core i5-8600K @ 3.6 Ghz
Grafická karta	GeForce GTX 970 4Gb VRAM, 1140 Mhz
Paměť	16Gb DDR4 3200Mhz
Úložiště	WD Black 2Tb, 7200rpm
Operační systém	Ubuntu 17.10

K programování byl vybrán jazyk Python, specificky byla zvolena jeho deep learning knihovna zvaná Keras. Daný nástroj vystavuje API nad frameworky TensorFlow, CNTK a Theano. Tato vysoká úroveň abstrakce umožňuje rychlý a efektivní návrh sítě, kdy lze vycházet z mnoha již naimplementovaných typů vrstev, aktivačních funkcí a učících mechanismů s možností kdykoliv potřebné třídy přetížit, či napsat vlastní. Využití GPU k učení těchto sítí je podporované, za předpokladu zkompilování knihovny CUDA přímo na konkrétní grafickou kartu.

```
model = Sequential()
# 100 vstupních uzlů, 32 neuronů v první vrstvě
model.add(Dense(32, activation='relu', input_dim=100))
# 1 výstupní neuron se sigmoid aktivací - True/False
model.add(Dense(1, activation='sigmoid'))
# Použití SGD s výchozím nastavením a vhodné loss funkce
model.compile(optimizer='SGD', loss='binary_crossentropy')
# Učení po 32 vzorcích a v celkem 10 etapách
model.fit(data, labels, epochs=10, batch_size=32)
```

Výpis 3: Ukázka definice jednoduchého binárního MLP v knihovně Keras

6.2 Dataset

Pro zjištění funkčnosti a efektivity výše uvedených neuronových sítí byla zapotřebí množina testovacích dat. Tu zpravidla výzkumníci získávají od antivirových společností nebo automatizovaným sběrem z vlastních sítí. V této práci je použit velmi známý a rozšířený veřejný dataset, jež byl publikován v roce 2015 v rámci Microsoft Malware Classification Challenge[25]. Tato sou-

též byla vyhlášena v roce 2015 společností Microsoft a jejím úkolem bylo přiřadit všem 10868 vzorkům jednu z devíti rodin/tříd.

Tabulka 3: Přehled datasetu

Třída	Rodina	Typ	Počet
1	Ramnit	Červ	1541
2	Lollipop	Adware	2478
3	Kelihos_ver3	Backdoor	2942
4	Vundo	Trojský kůň	475
5	Simda	Backdoor	42
6	Tracur	Trojský downloader	751
7	Kelihos_ver1	Backdoor	398
8	Obfuscator.ACY	Obfuskovaný malware	1228
9	Gatak	Backdoor	1013
	celkem		10868

Rozřazovaní mělo probíhat pomocí jakéhokoliv klasifikačního modelu naučeného na zcela odlišné sadě dalších 10868 malware, které byly předem přiřazeny správné třídy. Bohužel vzhledem k původnímu účelu těchto dat není k dispozici řešení, a proto je v rámci této diplomové práce celá originální validační část ignorována. Jako náhrada za ztracenou testovací množinu byla původní trénovací množina uměle rozdělena proporcionálním stratifikovaným náhodným výběrem[26], kdy 80% původního datasetu tvoří trénovací množinu a ze zbylých 20% vzorků byla vytvořena testovací sada.

6.2.1 Popis rodin v datasetu

Ramnit Tuto rodinu Microsoft klasifikoval jako červ a byla poprvé detekována v roce 2010. Tehdy se jednalo o jednoduššího červa, jež sloužil pouze jako médium pro přenos jiných hrozeb. Později došlo k vylepšování schopností rodiny. Autoři se inspirovali uniklým kódem trojského koně zvaného Zeus, který ukrádal finanční informace. Malware rovněž odcizoval údaje z FTP klientů nainstalovaných na cílovém zařízení a všechny je odesílal na server útočníků. Další přidanou funkcionalitou byla možnost ovládat malware vzdáleně, či dokonce získat přístup do napadeného zařízení. Cílený uživatel se mohl infikovat ze stránek, které se maskovaly jako charitativní organizace, pomocí externího úložného zařízení, jiného napadeného souboru a dalších. Pokud bychom tedy chtěli použít terminologii sekce 3.1, můžeme tuto rodinu označit jako spyware, trojský kůň, virus, červ, botnet i backdoor. „Popularita“ tohoto malware vyvrcholila v roce 2016, kdy se udávalo, že tento botnet obsahoval 3,2 miliónu zařízení a z tohoto důvodu byl zaměřen organizací Europol, která pod záštitou Evropské unie bojuje s organizovanou trestní činností. Ačkoliv byla tato operace označena za úspěšnou, dodnes je tato rodina aktivní a nebezpečná[27][28].

Lollipop Jedná se o zcela typický příklad adware, jenž infikovaným uživatelům zobrazuje „vyskakující“ reklamy. Tento program se do cílového zařízení nainstaluje jako přídavek k nějakému určitému „chtěnému“ software. Uživateli je při instalaci dána volba tento „doplněk“ neinstalovat, avšak většina uživatelů si nepročítá instalační instrukce a rovnou volí tlačítko další i v případě tohoto potencionálně nechtěného programu. Malware se nijak neskrývá ani se nesnaží zamezit jeho odstranění, dokonce nabízí separátní „uninstaller“[29].

Kelihos Kelihos je zástupcem kategorie hybridních botnetů. Napadené zařízení je začleněno do sítě botů, kde každý tento stroj přijímá práce od útočníka. Hybridní architektura je dána použitím centrálních serverů i *peer-to-peer* sítě. Nejčastější škodlivou aktivitou bylo nalezení a odcizení Bitcoin peněženek, či rozesílání nevyžádaných emailů.[30]

Vundo Typický zástupce trojského koně kdy malware se do cílového zařízení dostává předstíráním benigního souboru. Příkladem mohou být emailové přílohy vypadající jako obrázky (image.png.exe), kodeky, generátory klíčů distribuované v peer-to-peer sítích atp. Samotný program je pak schopen upravovat navštěvované webové stránky a přidávat do nich reklamy, modifikovat výsledky vyhledáváčů, měnit grafické nastavení operačního systému a mnoho dalších[31].

Simda Tato poměrně komplikovaná rodina byla nejaktivnější v letech 2012 - 2015. Po mezinárodní kolaboraci, do které se zapojilo mnoho různých organizacích zaměřující se na obranu proti kyberzločinu jako Kaspersky, Trend, Microsoft, FBI, CDI a mnoho dalších, byl zastaven botnet o velikosti přes tři čtvrtě milionu zařízení. Tento malware po nainstalování zkontroloval, zda-li neběží na virtuálním zařízení, zkopíroval se do několika složek, upravil registry, aby umožnil opětovné spuštění programu při startu systému a otevřel příslušné porty v případném Microsoft Firewallu. Tento backdoor je ukázkovým příkladem malware tvořícího Command and Control strukturu. Mezi příkazy, které mohl útočník na cílovém zařízení spouštět, patří například vynucený restart, smazání specifických souborů, změna nastavení, zahájení shromažďovaní citlivých informací a jejich odeslání. Z databází antivirových společností není zřejmé, jak se rodina šířila[32].

Tracur Jedním z účelů tohoto malware je přesměrovávat výsledky specifických vyhledávacích stránek na jiné stránky zobrazující reklamu, která autorům vydělávala peníze. Dalším účelem pak bylo vytvoření zadních vrátek a tajného stahování dalších potencionálně škodlivých souborů[33].

Obfuscator Na rozdíl od ostatních tříd, tato kategorie není malware rodinou. Do této skupiny patří malware, který byl jakýmkoliv způsobem obfuskován, jak blíže popisuje sekce o obraně proti statické analýze.

Gatak Jedná se o spyware, který se do cílových zařízení dostával pomocí programů generujících licenční klíče, které se používaly pro aktivaci komerčních produktů bez nutnosti placení.

Nakažený stroj pak poskytoval útočníkovi „zadní vrátka“ a shromažďoval citlivé informace. Zajímavostí této rodiny bylo použití steganografie k získání dalších instrukcí. Malware si stahoval na pohled zcela normální obrázky z předem určených webových stránek, které pak dešifroval a získal patřičné informace[34].

6.2.2 Reprezentace malware v datasetu

Název každého malware byl nahrazen unikátním 20 znaků dlouhým *hashem*. Samotný vzorek je tvořen párem dvou souborů. Ten menší z nich má koncovku *.bytes a obsahuje paměťovou adresu a samotná data v podobě hexadecimálních číslic. Druhý, obsáhlejší soubor končí písmeny *.asm a kromě adresy a hexadecimálních dat zahrnuje také název sekce, funkce včetně parametrů a případné komentáře. Oba soubory byly vygenerovány disassemblerem IDA Pro[35].

Níže uvedené kódy zobrazují stejný malware v obou dostupných reprezentacích. Asm soubor byl pro nedostatek místa a lepší čitelnost zbaven komentářů.

```
1 00401820 55 8B EC 6A FF 68 80 84 42 00 64 A1 00 00 00 00
2 00401830 50 83 EC 0C 53 56 57 A1 60 D2 43 00 33 C5 50 8D
3 00401840 45 F4 64 A3 00 00 00 00 89 65 F0 8B F9 89 7D EC
4 00401850 8B 45 08 8B F0 83 CE 0F 83 FE FE 76 04 8B F0 EB
5 00401860 22 8B 5F 18 B8 AB AA AA AA F7 E6 8B CB D1 E9 D1
6 00401870 EA 3B D1 73 0E B8 FE FF FF FF 2B C1 3B D8 77 03
7 00401880 8D 34 19 8D 4E 01 6A 00 51 C7 45 FC 00 00 00 00
```

Výpis 4: Ukázka .bytes souboru

Jak bylo naznačeno výše, každý řádek začíná 4 byty dlouhou adresou v hexadecimálním tvaru. Po jedné mezeře pak následuje 16 bytů dat rovněž v šestnáctkové soustavě, které jsou také odděleny mezerami. Následující řádek opět začíná adresou tentokrát inkrementovanou o právě zobrazených 16 bytů (10 v hexadecimální soustavě).

```
.text:00401820 55                    push    ebp
.text:00401821 8B EC                 mov     ebp, esp
.text:00401823 6A FF                 push    0FFFFFFFFh
.text:00401825 68 80 84 42 00        push    offset loc_428480
.text:0040182A 64 A1 00 00 00 00     mov     eax, large fs:0
.text:00401830 50                    push    eax
.text:00401831 83 EC 0C              sub     esp, 0Ch
.text:00401834 53                    push    ebx
.text:00401835 56                    push    esi
.text:00401836 57                    push    edi
```

Výpis 5: Ukázka .asm souboru

Druhý soubor již není tak pravidelný, ale každý řádek obsahuje různý počet informací. Jednotlivé řádky jsou vždy zahájeny názvem sekce a adresou. Po tomto úvodu následuje arbitrární počet bytů (nejvýše však 7). V uvedeném případě můžeme vidět, že hexadecimální číslice 56 vyjadřuje operaci *push*, jejíž argumentem je registr *ebp*. Dobré je si povšimnout, že znak separující hexadecimálních dat a přeložené operace není jednoznačný. Po prvních třech bytech následuje namísto mezery tabulátor jinak jsou použity klasické mezery. Pokud jsou na řádku zaplněny všechny dostupné pozice, je oddělovacím znakem mezera. Došlo-li ke skrytí určitých dat z důvodu nedostatku místa, je separátorem znaménko „+“. Detekovat tuto strukturu je nutné pro správné parsování dat. Bohužel se zadavatel soutěže neobtěžoval dataset jakkoliv dokumentovat.

6.3 Obecný postup při analýze

Dříve než-li budou popsány jednotlivé analýzy, je vhodné vylíčit obecné postupy v nich využívané.

V rámci praktické práce byly zvoleny čtyři poměrně unikátní pohledy na danou problematiku. U každého je důkladně popsán postup pro získání vstupních dat. Následně je zpravidla prováděno vyhledání architektury a zjištění optimálních hyperparametrů. Vzhledem k dlouhým učícím dobám u některých sítí je toto prohledávání vykonáváno ručně s pouze omezeným datasetem (např. pouze 500 vzorků z každé třídy). Pokud došlo k omezení datasetu, byla ještě provedena opětovná optimalizace parametrů s plnou verzí datasetu, avšak v daleko nižší míře. Výsledná konfigurace je přehledně vyobrazena jednoduchým schématem a doplňující tabulkou, která pro každou vrstvu sítě udává její parametry, množství naučitelných vah a finálně její výstup.

Při samotném procesu učení dojde na konci každé epochy k aktuálnímu otestování sítě a uložení všech příslušných výsledků. Výstupem je tedy čtveřice hodnot - ztráta a přesnost trénovací sady a stejný pár pro testovací sadu. Pro úplnost, **ztráta sítě** je zde definována jako suma všech jednotlivých ztrátových funkcí v rámci jedné epochy a **přesností** budu po dobu této práce uva-

žovat poměr (obvykle v procentech) mezi správně klasifikovanými případy a celkovým počtem příkladů. V rámci analýzy bude pro nejúspěšnější výsledky vykreslen graf těchto čtyř zmíněných hodnot. Pro znázornění finálního stavu klasifikace testovací sady bude použita takzvaná konfuzní matice. Řádky této matice představují pravdivé třídy, tedy tu skupinu do které chceme aby síť vzorek klasifikovala a sloupce jsou predikované třídy, tedy skupiny jež určila samotná síť. Ideálně by měly být všechny vzorky situovány na diagonále, což by představovalo bezchybnou klasifikaci. Vzhledem k rozdílnému množství vzorků v každé třídě je tato matice zobrazena vždy ve dvou reprezentacích. Kromě klasické formy obsahující absolutní číselné hodnoty je vždy k dispozici i procentuální podoba, kdy každá hodnota udává, kolik procent ze vstupních dat bylo zařazeno do konkrétní třídy.

Výstupem každého učení je soubor CSV, jenž obsahuje ztráty a přesnosti na obou sadách pro každou epochu, dále soubor obsahující všechny naučené váhy včetně odpovídající architektury a parametrů a také vizualizace zmíněné v předchozím odstavci. Díky tomuto lze velmi jednoduše zpětně vyhledat již provedený experiment a použít jej znovu například na jiných datech.

6.4 Analýza bitových sekvencí

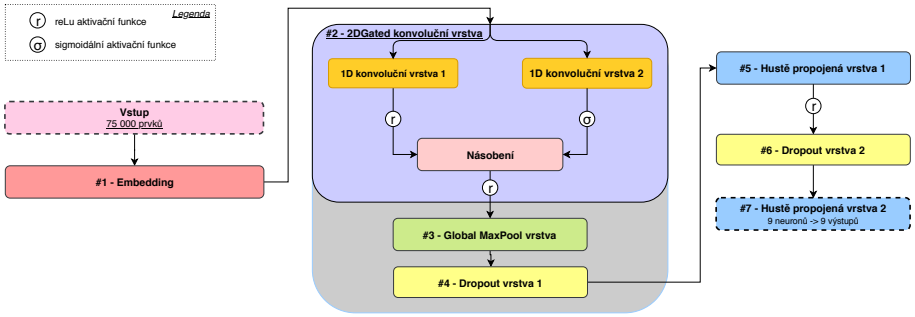
Tato analýza je silně inspirována prací popisující rozpoznávání malware čtením kompletní sekvence bytů. K tomuto účelu byla použita jednodimenzionální konvoluční síť, konkrétně typ Gated Convolutional Network.

Originální soubory datasetu obsahují kromě samotné bitové sekvence také adresy dat, jež v tomto případě nepotřebujeme. Jejich odstraněním dojde k poklesu velikosti zpracovávaného souboru a mírnému ušetření výpočetního času při učení. Zbývající byty jsou zapsány v hexadecimální reprezentaci. Kromě očekávaných hodnot 00 až FF, se však v souborech vyskytují byty označené „??“. Ačkoliv není přesně zdokumentována příčina těchto artefaktů, pravděpodobně jde o nepovedenou *disassembly* vlivem obfuskace nebo sekvence dat, která nemůže být disassemblerem jednoznačně namapována na známou instrukci. Jelikož neuronová síť očekává dekadické numerické hodnoty, byla hexadecimální reprezentace převedena do decimální soustavy, tedy na číslice 0 až 255. Byty s neznámou hodnotou byly nahrazeny číslem 256, aby došlo k zachování informace, která teoreticky může pomoci při klasifikaci. Výsledná data byla zapsána do zvláštních textových souborů s koncovkou *.mksbytes*. Tato operace trvá na zvoleném hardware řádově desítky minut, dojde však ke značnému ulehčení parsování souboru při budoucím učení sítě. Jelikož se jedná o desítky GB dat, byl rovněž proveden experiment s ukládáním dat v alternativních formátech pomocí knihoven *pickle* (ukládání přímé reprezentace dat v pythonu pro rychlejší načítání), *ultraJSON* (podobně jako pickle, ale navíc s komprimací) atp. Výhody však nebyly dostatečně významné, a proto se pro tuto analýzu budou využívat soubory *.mksbytes*.

Jak již bylo zmíněno v úvodu, inspirací pro tuto analýzu byla práce „Malware Detection by Eating a Whole EXE“[23]. Jejich model nazvaný „MalConv“ byl uzpůsoben binární klasifikaci

miliónů vzorků o maximální velikosti 2Mb, nicméně nebyl důvod podezřívat, že by obdobný postup nemohl být aplikován na více-třídý klasifikační problém. Zmíněná práce velmi dobře popisuje všechny vrstvy včetně jejich parametrů, což se stalo dobrým výchozím bodem.

První konfigurace vzhledově kopíruje „MalConv“. O extrakci rysů se stará tzv. „Gated Convolution Layer“. Ve své podstatě to jsou dvě parametricky shodné konvoluční vrstvy, které paralelně přijímají stejný vstup. Každá z těchto konvolucí má své váhy a jejich hlavní odlišností je použitá aktivační funkce, kdy jedna z nich používá sigmoidální aktivaci, přičemž druhá aktivace je arbitrární, zpravidla reLu. Výstupy z této dvojice se vynásobí mezi sebou tzv. Hadamardovým součinem[36] - tedy každá se vynásobí s prvkem druhého vektoru na stejné pozici v druhém vektoru. V literatuře se tomuto produktu velmi často říká „Elementwise multiplication“ neboli „Multiplikace po složkách“. Po gated convolution vrsvě následuje globální max pool vrstva. Zde dojde k získání jediné maximální hodnoty z každého filtru, a tedy k poměrně extrémní redukci informací. Dle práce „Network in Network“[37],ve které byla tato struktura poprvé zmíněna, se tato vrství chová podobně jako hustě propojená vrstva s vysokou odolností proti přeučení a její výstup je možno v extremních případech použít přímo spolu se softmax aktivací ke klasifikaci. V tomto případě je však použita ještě jedna hustě propojená vrstva s učitelnými vahami.



Obrázek 7: Architektura Gated Convolution sítě pro bytové sekvence

Po ustanovení obecné struktury sítě bylo zapotřebí najít vhodné parametry pro tuto specifickou klasifikační úlohu. Vlivem velkého množství vstupních dat a jejich embeddingu vyžaduje síť velmi mnoho místa v paměti a její učení trvá v řádech hodin. Hledání pouze malé kombinace vstupních parametrů by na celkovém datasetu znamenalo úlohu s časovou náročností v řádech dnů. Z tohoto důvodu došlo k omezení počtu vzorků datasetu pomocí stratifikovaného výběru. Jelikož i tento postup se ukázal být časově velmi náročným, byly zmenšeny vstupní vektory na maximálních 500kB a trénované vzorky byly vybírány tak, aby se do tohoto omezení zcela vešly. Jelikož tak nastalo poměrně veliké vychýlení od původního zadání problému, byly výsledky tohoto prohledávání brány s rezervou. Nalezená optima se oproti výchozímu bodu převzatého z původní práce příliš nelišila. Zmenšením parametru stride v konvolučních vrstvách došlo k mírnému navýšení výpočetního času, avšak na našem modifikovaném zadání vykazovalo postupné zmenšování zlepšení přesnosti v řádu jednotek procent.

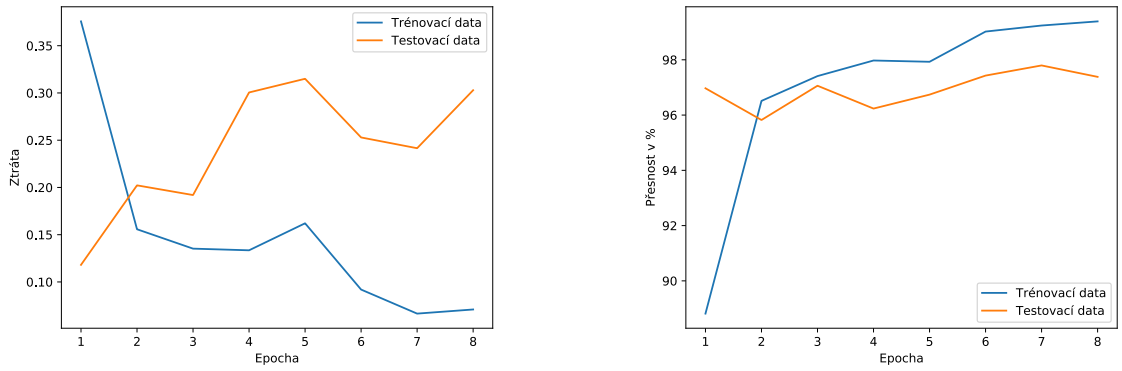
Parametry použité k prvnímu učení:

Tabulka 4: Parametry prvního učení Gated 1D CNN na bytech

#	Název vrstvy	Parametry	Učitelné váhy	Výstup
0	Vstupní vr.	-	-	5 000 000 B
1	Embedding vr.	8 dimenzí	2 056	5 000 000 x 8
2	Gated 1D konv. vr.	128 filtrů o délce 500 (400 s.)	512 128	12 499 x 128
3	Globální MaxPool vr.	-	-	128
4	Dropout vr. 1	zahodí 0%	-	128
5	Hustě propojená vr. 1	128 neuronů	16 512	128
6	Dropout vr. 2	zahodí 40%	-	128
7	Hustě propojená vr. 2	9 neuronů	1 161	9

I přesto, že byl kladen značný důraz na kvalitu předzpracování, aby došlo k redukci zbytečných operací při učení, trvala každá epocha přibližně jednu hodinu. Samotná validace, která je prováděna na konci každé epochy trvala až 10 minut. Naštěstí u modelu dojde rychle k přeučení a stačí v průměru pouhých 9 epoch k nalezení maximální přesnosti. Už při první epoše dokázala síť predikovat správnou třídu testovací množiny s více než 94% přesností. Celkově veškeré učení a validace v devíti epochách vyžadovaly téměř deset hodin času na GPU. Při zpětné kontrole a časování jednotlivých složek bylo zjištěno, že nejkritičtějším místem je proces čtení z disku.

Při vylepšeném algoritmu (užitím vlastní implementace knihovny numpy) pro čtení z disku, došlo k redukci času potřebného k celému procesu na necelých 8 hodin.



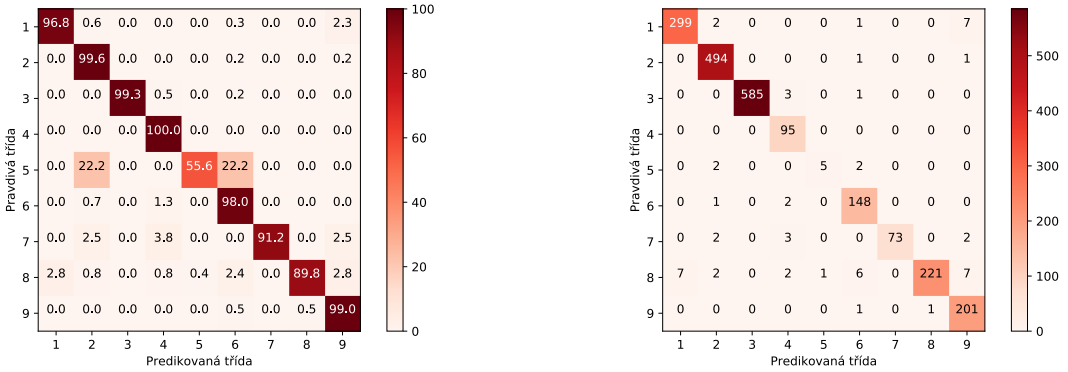
(a) Ztráta při učení

(b) Přesnost při učení

Obrázek 8: Průběh učení

Na předcházejícím obrázku lze vidět, že se síť skutečně učí velmi rychle, už při první iteraci, je síť naučená na trénovací sadě s přesností 88% a při osmé už 99,5%. Relativní nevzhlednost grafu ztrát způsobuje poměrně vysoká míra učení, při snížení této hodnoty by se síť pravděpodobně učila méně chaoticky, bohužel zvýšila by se také doba celkového procesu, což je při takto dlouhých

epochách velmi nechtěné. I přesto, že jsme vlivem pomalého učení nemohli důsledně prověřit vliv parametrů na výkon, dokazuje síť velmi zdařilých výsledků. Nejlepší výsledek na testovací sadě byl nalezen v sedmé epoše a dosahuje **97,8 %** úspěšnosti se ztrátou **0,2415**, přičemž přesnost na trénovací sadě byla **99,4%** se ztrátou **0,0665**.



(a) Procentuální reprezentace

(b) Absolutní reprezentace

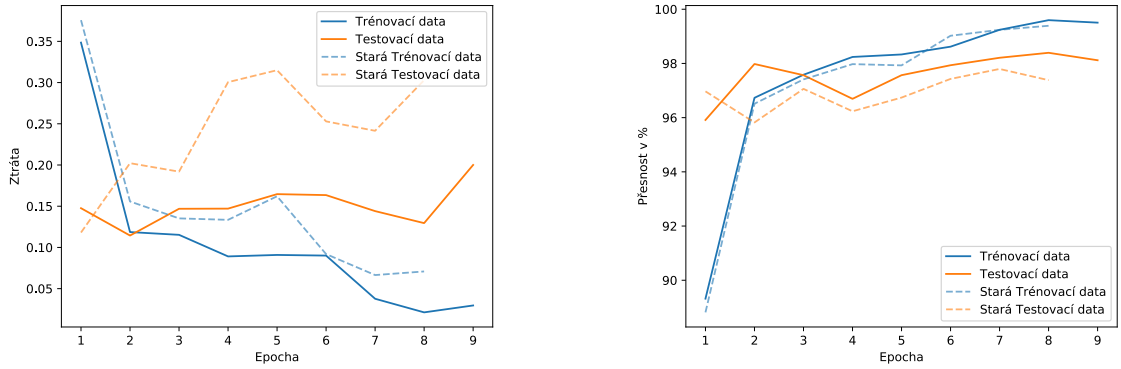
Obrázek 9: Konfuzní matice analýzy 150x150 obrázků

Dle originální práce dosahuje síť optimálních výsledků při užití velkých filtrů a skoků, je však možné, že v tomto specifickém případě je důležitější vyšší granularita. Druhý pokus tedy testuje užití filtrů o téměř pětinovém rozlišení se stride o poloviční velikosti, tedy sousední neurony budou sdílet polovinu vstupů. Výsledné nastavení popisuje následující tabulka.

Tabulka 5: Parametry prvního učení Gated 1D CNN na bytech

#	Název vrstvy	Parametry	Učitelné váhy	Výstup
0	Vstupní vr.	-	-	5 000 000 B
1	Embedding vr.	8 dimenzí	2 056	5 000 000 x 8
2	Gated 1D konv. vr.	128 filtrů o délce 128 (64 s.)	131 120	78 124 x 128
3	Globální MaxPool vr.	-	-	128
4	Dropout vr. 1	zahodí 0%	-	128
5	Hustě propojená vr. 1	128 neuronů	16 512	128
6	Dropout vr. 2	zahodí 40%	-	128
7	Hustě propojená vr. 2	9 neuronů	1 161	9

V tomto případě tedy dochází k mnohem většímu počtu konvolucí, což má za následek více hodnot, ze kterých se získává maximum. Jelikož také došlo ke zmenšení filtru máme v tomto případě mnohem méně trénovatelných vah. Tyto dvě skutečnosti mírně zrychlily průběh učení na výsledných 7,17 hodin při celkových devíti epochách, což znamená 48 minut na jednu epochu.

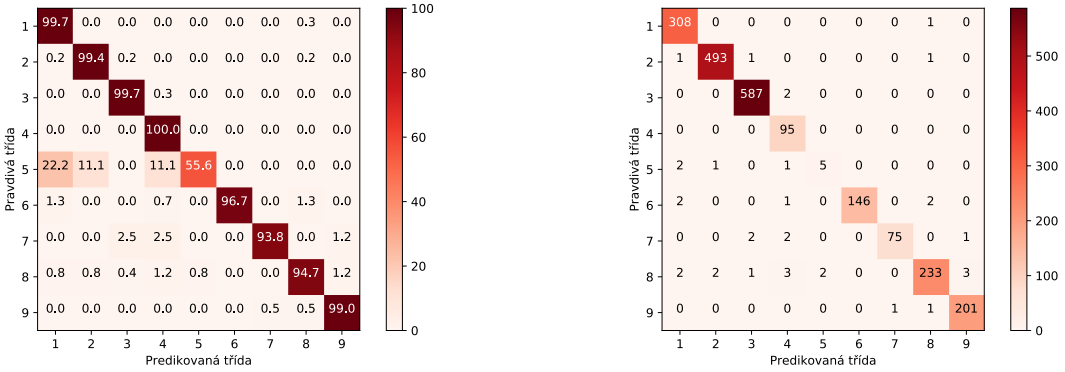


(a) Ztráta při učení

(b) Přesnost při učení

Obrázek 10: Průběh učení bitové sekvence při užití menších filtrů

Čas nebyl jediné u čehož došlo ke zlepšení. Na předchozích grafech lze vidět nové hodnoty, ty předchozí jsou zaznačeny průhlednější přerušovanou čarou. Zlepšení lze vidět ve ztrátě i přesnosti obou sad. Nejlepší výsledek byl nalezen v osmé epoše, kdy bylo správně klasifikováno **98,4%** testovacích dat.



(a) Procentuální reprezentace

(b) Absolutní reprezentace

Obrázek 11: Konfuzní matice bitové sekvence při užití menších filtrů

6.5 Analýza bitového obrazu

Tato analýza je postavena na premise, že někteří autoři malware mnohdy vytvoří varianty svých programů se stejným jádrem, přičemž upraví pouze pomocný kód, nebo pozmění obfuskaci, či pozpřehazují již stávající sekce programu. Pokud si takový škodlivý kód vizualizujeme, měli bychom v jedné specifické rodině být schopni najít podobné struktury. Není tedy nutné porozumět jednotlivým hodnotám, dokonce vidět detailní náhled programu, stačí analyzovat černobílý

obrázek vzniklý z originálních bytů kódu, který byl zmenšen na určitou, překvapivě malou velikost.

Předzpracování vychází z dat získaných v rámci minulé analýzy, tedy ze souborů končících „.mksbytes“. Ty již obsahují hodnoty 0 - 256. Jelikož je cílem získat černobílý obraz, jehož intenzita je udána 8 bitovou hodnotou, je nutné číslice 256 snížit na 255, tedy zcela černý pixel. Tímto krokem bohužel ztrácíme informaci původně vyjádřenou symboly „??“. V tuto chvíli máme jedno-dimenzionální vektor o variabilní délce. Dalším krokem je omezit šířku tak, aby došlo k „zalomení“ obsahu na nový řádek a vznikla tak dvou-dimenzionální struktura. Tato šířka může být fixní, nebo se může měnit v závislosti na celkové velikosti sekvence, jak bylo užito v citované práci a nakonec i v této analýze. Konkrétní šířky znázorňuje následující tabulka.

Tabulka 6: Výpočet šířky obrázku

Velikost souboru (B)	**Šířka obrázku (px)**
0 - 9 999	32
10 000 - 29 999	64
30 000 - 59 999	128
60 000 - 99 999	256
100 000 - 199 999	384
200 000 - 499 999	512
500 000 - 999 999	768
1 000 000 - inf	1024

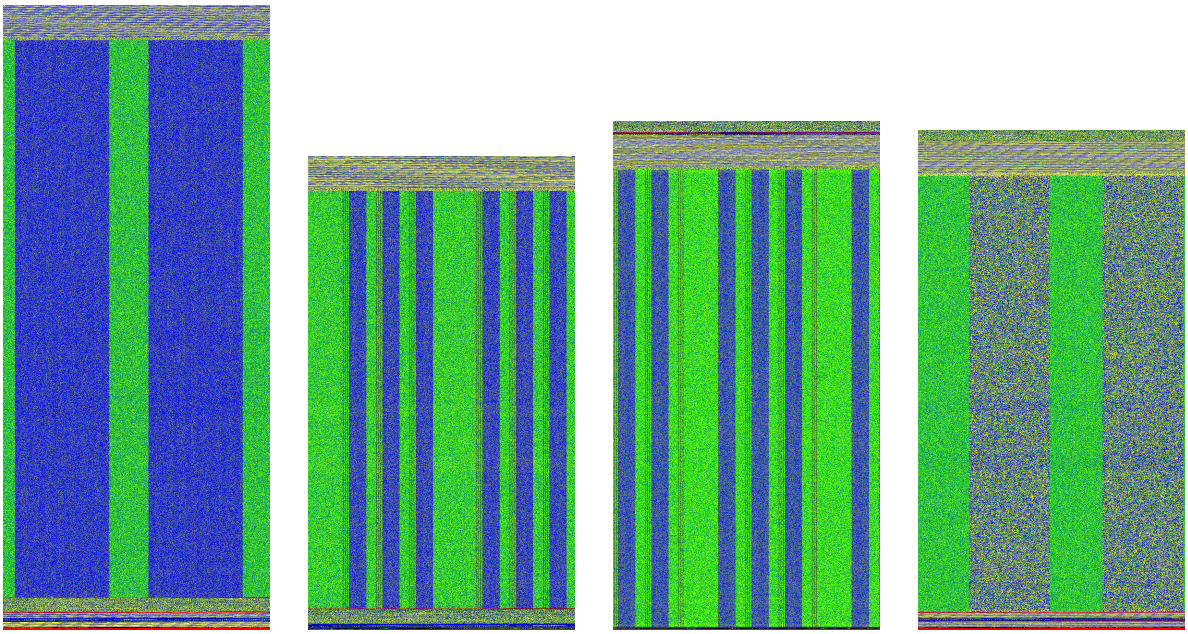
Vzniklá dvou-dimenzionální struktura je uložena jako soubor ve formátu *.PNG*. Aby byl vstup zpracovatelný konovoluční sítí, zbývá vyřešit poslední překážka, kterou je variabilní velikost obrazu. Ve vědecké práci, ze které tato analýza vychází, bylo použito zmenšení všech reprezentací na fixní rozměry 32x32 (čímž nutně došlo k vizuální „deformaci“). V rámci tohoto předzpracování byl využit stejný princip, ale ve více rozměrech, které byly mezi sebou porovnány. Celkově byly vygenerovány 4 samostatné datasety s dimenzemi 32x32, 64x64, 150x150 a 300x300.

Čistě pro účely vizualizace byl vytvořen ještě jeden skript, který využíval více barevných kanálů. Inspirací byl program s názvem PortEx vytvořený Katja Hahnem v rámci své diplomové práce[39]. Jednou z vlastností tohoto programu je skvělá vizualizace, která umožňuje lepší znázornění entropie, rozložení sekcí a hlavně zobrazuje stejnou reprezentaci jako náš právě vytvořený dataset, s tím rozdílem, že nepoužívá 255 odstínů šedi, ale 5 fixních barev. Pro strojové vidění je zvolený typ zobrazení mnohem méně použitelný, jelikož dojde ke ztrátě informace a zvětšení dimenze dat. Naopak pro lidské oko je tento způsob přijatelnější, protože malé rozdíly v intenzitě pixelu lze špatně vidět, ale různé barvy vidíme snáze.

Stejným principem tedy funguje i poslední skript. Pixely s nulovou hodnotou zobrazuje černě, číslo 255 reprezentuje bílou barvou, hodnoty 1 až 31 jsou zvýrazněny zeleně (neviditelné ASCII charaktery), 32 - 127 jsou vybarveny modře a nakonec číslice 128 - 254 jsou znázorněny žlutou

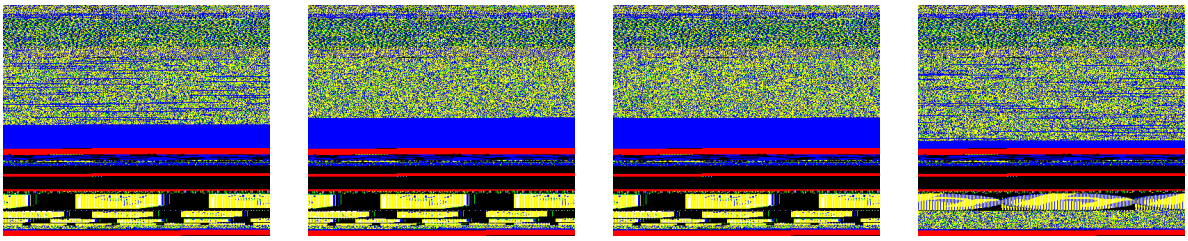
barvou. Soubor mksbytes obsahuje i hodnotu 256, která reprezentuje byty s neznámou hodnotou „??“ a která musela být v klasifikační datové množině nahrazena bílou barvou. V tomto skriptu není takové omezení, a proto je tato hodnota ve výsledném obrázku zaznačena červenou barvou.

Mnohé takto vizualizované vzorky jsou pro lidské oko pouhým šumem. U některých tříd však můžeme nacházet určitý vzorec. Obrazy rodiny Kelihos_v3 (#7) jsou vždy z asi 90% pokryty červenou barvou. Na mnohých vizualizacích vzorků, jež byly obfuskovány (#8) můžeme zpozorovat navzájem velmi podobné struktury objevující se v posledních sekcích souboru. Některé soubory se dokonce liší pouze v jednotlivých částech. Těchto podobností lze nalézt poměrně mnoho. Už jenom tímto zjištěním se tato cesta počínání zdá být velmi perspektivní.



(a) Vzorek #1 (b) Vzorek #2 (c) Vzorek #3 (d) Vzorek #4

Obrázek 12: Vizualizace vybraných vzorků třídy Lollipop (#2)



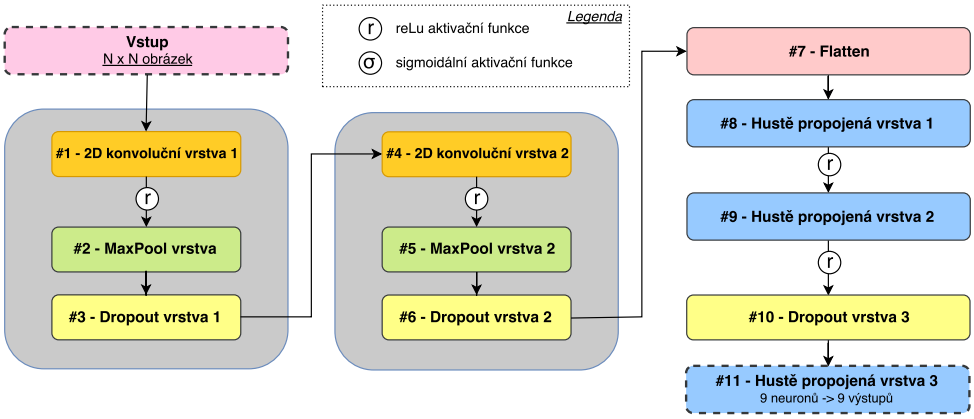
(a) Vzorek #1 (b) Vzorek #2 (c) Vzorek #3 (d) Vzorek #4

Obrázek 13: Vizualizace vybraných vzorků třídy Obfuscator.ACY (#8)

V oboru neuronových sítí se pro analýzu dvou-dimenzionálních obrázků nejčastěji používají konvoluční sítě. Ačkoliv architektur, ze kterých by se v tomto případě mohlo vycházet existuje

velmi mnoho, pro jednoduchost byla použita ta nejstarší a nejobecnější z nich popsaná v sekci „Organizace vrstev do sítě“. V této architektuře nejprve dochází k extrakci rysů pomocí několikrát se opakující konvoluce a maxpoolu. Tyto rysy poté přechází do několika hustě propojených vrstev, kde dochází k samotné klasifikaci. Abychom zabránili přeučení, byla tato architektura obohacena o dropout vrstvy hned po maxpool vrstvách a další po hustě propojených vrstvách.

Pro počáteční vyhledání vhodného uspořádání sítě byly použity obrázky o rozměrech 64x64 a optimalizátorem SGD. První vyzkoušená konfigurace obsahovala pouze jednu konvoluční vrstvu, max pool a výstupní hustě propojenou vrstvu. Postupným přidáváním vrstev byla nalezena optimální konfigurace, znázorněná na následujícím obrázku.



Obrázek 14: Architektura sítě 2x2dC+2xD

Jelikož učení takto rozměrných obrázků vyžadovalo v průměru pouze 15 minut, mohla být vykonána optimalizace užitím grid search metody. Tímto způsobem byly vyzkoušeny různé velikosti dávek, optimalizátory, konstanty učení a hyperparametry vrstev. Ve konečné podobě byl použit optimalizátor SGD s Nesterovovou akcelerací, konstantou učení 0,01 a velikostí dávky 16. Hyperparametry jsou popsány přehledněji v tabulce níže.

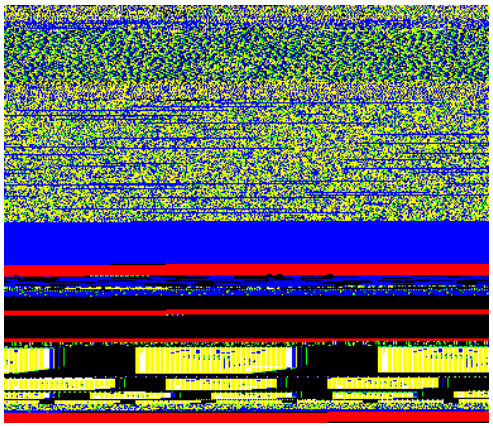
Tabulka 7: Finální parametry Gated 1D CNN pro ASM

#	Název vrstvy	Parametry	Učitelné váhy	Výstup
1	Embedding vr.	8 dimenzí	7 280	75 000 x 8
2	Gated 1D konv. vr.	32 filtrů o délce 16	8 256	74 985 x 32
3	MaxPool vr.	128 znaků	0	585 x 32
4	Flatten vr.	-	0	18 720
5	Dropout vr. 1	zahodí 10%	0	18 720
6	Hustě propojená vr. 1	256 neuronů	4 792 576	256
7	Hustá propojená vr. 2	128 neuronů	32 896	128
8	Dropout vr. 2	zahodí 40%	0	128
9	Hustě propojená vr. 3	9 neuronů	1 161	9

V předzpracování byly vytvořené 4 datasety lišící se rozměry obrazů. V původní práci, která popisovala tuto analýzu byl zvolen rozměr 32x32. Možným důvodem pro takto malé rozlišení mohl být tehdejší výpočetní výkon, který se za 7 let od vydání zmíněné práce podstatně zlepšil. Všechny rozměry byly porovnány na stejné architektuře se stejnými parametry. Tento způsob tedy nutně neudává nejlepší možnou přesnost pro daný rozměr, jelikož pro menší rozlišení by se měly zmenšit i rozměry kernelu konvoluční vrstvy, aby došlo k zachování extrakcí rysů z podobných oblastí a obdobně postupovat i při užití obrázků s větším rozlišením.

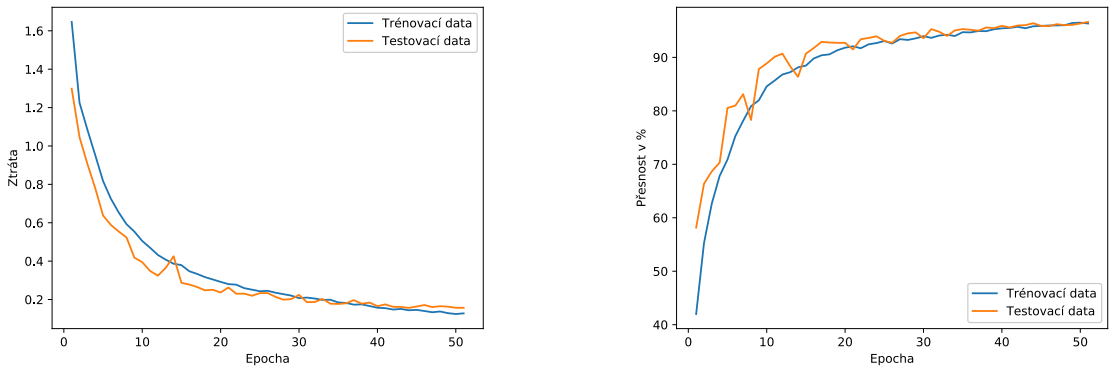
Velikost	Nej. přesnost	Celková doba
32 x 32	96,5%	6,2 m
64 x 64	97,1%	12,9 m
150 x 150	97,3%	65,2 m
300 x 300	-%	180+ m

Tabulka 8: Shrnutí učení více velikostí



Obrázek 15: Učení více velikostí

Finální učení na datasetu tvořeného obrázky o dimenzích 64x64 trvalo pouhých 357 sekund, rozdělených do 51 epoch. Síť se po této době ustálila na ztrátě **0,1279**, což odpovídá přesnosti **96,34%**. Po ukončení trénování nastala finální aktivace za použití testovací množiny dat. Výsledná ztráta této sady byla **0,1569** s přesností **96,64%**. Stav těchto hodnot v každé epoše je znázorněn na následujícím obrázku.

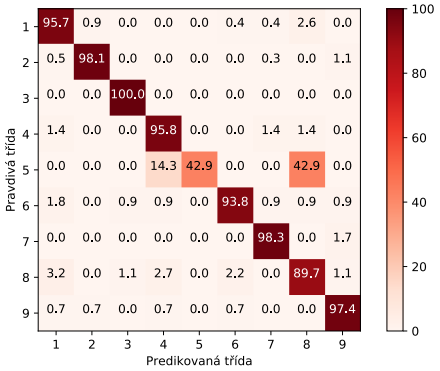


(a) Ztráta při učení

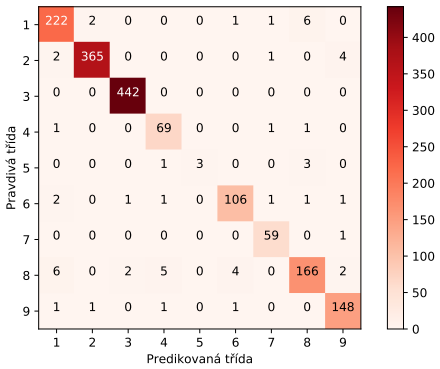
(b) Přesnost při učení

Obrázek 16: Průběh učení 64x64 obrázků

Při pohledu na konfuzní matici níže, lze zpozorovat nejkritičtější chyby v klasifikaci. Zcela očekávaně je problém v rozpoznání 5. třídy, která obsahuje pouhých 42 vzorků, přičemž k učení jich bylo použito pouze 35. Další problematickou třídou je 8. rodina, což je skupina, na které byla použita obfuskace. Tyto chyby jsou odůvodnitelné ztrátou informace o špatně přeložených bytech, jež byla zmíněna v sekci o předzpracování dat.



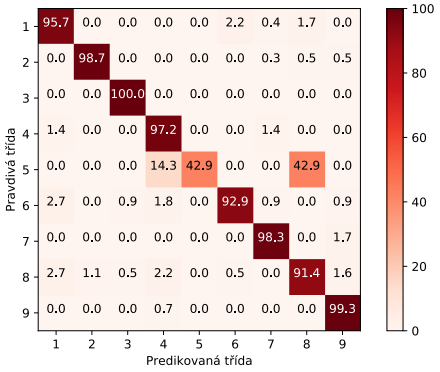
(a) Procentuální reprezentace



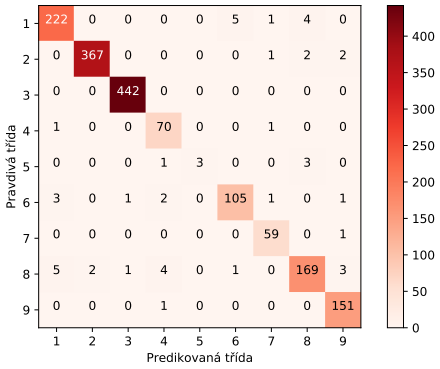
(b) Absolutní reprezentace

Obrázek 17: Konfuzní matice analýzy 64x64 obrázků

Při použití větších zdrojových obrázků se situace mění pouze marginálně. Konkrétně při použití velikostí 150x150, což je přibližně 5,5 násobné zvětšení, dojde ke zlepšení finální situace o cca 0,5% na **97,12%**. Při dalším zvětšování je zisk stále nižší, zatímco výpočetní náročnost roste kvadratickou mírou.



(a) Procentuální reprezentace



(b) Absolutní reprezentace

Obrázek 18: Konfuzní matice analýzy 150x150 obrázků

6.6 Analýza čítačů

Na rozdíl od bytového souboru, obsahuje ASM ekvivalent mnohem více informací. Stejně jako v předchozím typu, i tento soubor obsahuje bytovou reprezentaci s adresou. Navíc zde nalezneme i názvy jednotlivých sekcí, u nichž lze pomocí jejich adres lze jednoduše odvodit délku. Samozřejmostí jsou operační instrukce a argumenty těchto funkcí. Dalším důležitým zdrojem informací jsou komentáře, které mimo jiné upřednostňují vlastnosti sekcí, importované knihovny a funkce, či naopak exportované artefakty. Cílem této analýzy je sestavit vektory udávající, kolikrát se určitý příznak objevuje v každém ze souborů a z těchto fixních vektorů naučit vícevrstvou síť skládající se z hustě propojených vrstev.

Prvním nutným krokem je důkladné zpracování ASM souboru. Bohužel jelikož neexistuje žádná dokumentace k vygenerovaným souborům, vznikalo parsováním velmi mnoho chyb. Jelikož podstatou této práce není samotné zpracování rozsáhlých dat, byl použit parsovací algoritmus převzatý z GitHub repositáře autora Oleksandra Lysenka[38]. Tento skript nebyl bezchybný a vykazoval mnoho zbytečných iterací nad daty než bylo nutné. redukci nutnosti přemýšlet nad strukturou zdrojových dat, separací jednotlivých informací atp.

Soubor je čten po řádcích. Na začátku je přečten název momentální sekce a adresa aktuálního řádku. V případě, že název sekce se změnil oproti předchozímu řádku, dojde k odečtení aktuální adresy od poslední zapamatované, čímž efektivně spočítáme délku bývalé sekce. Zapamatovanou adresu nahradíme tou aktuální. Dále dojde k prohledání komentáře řádku, tedy hodnoty za znakem „!“, v komentářích se mohou vyskytovat informace o importovaných knihovnách, funkcích, exportech a mnoho dalších. Jako poslední operace dojde k parsování samotných instrukcí, přičemž je zapamatován pouze název a veškeré argumenty funkcí jsou ignorovány. Po přečtení všech řádků dochází ke generování n-gramů, tedy počítání všech možných n-tic ze sekvence operací. Pro tuto analýzu byly použity samostatné funkce, dvojice a trojice. Počet kombinací při použití více členů se velmi rychle zvyšuje.

Všechny čítače jsou zapsány do příslušných *.CSV* souborů, kde řádky představují jednotlivé vzorky a sloupce počet výskytů daného rysu v tomto souboru. Zvlášť se ukládají informace o sekcích, importovaných knihovnách, funkcích atp. Tímto způsobem lze lehce ovlivňovat, ze kterých informací se síť bude učit. Pokud bychom chtěli použít všechny nalezené informace, měl by vstupní vektor do sítě délku 1,157,469 příznaků. Ačkoliv jsou neuronové sítě obecně odolné vůči nadbytečným informacím, přidávají tyto skutečnosti zbytečně na výpočetní složitosti. Proto ještě došlo ke statistické analýze vyextrahovaných hodnot a redukci příznaků.

Proces využíval matematického vzorce, který se běžně užívá k identifikaci odlehlých pozorování. Těmi jsou data, jež se významně liší od ostatních. V klasické statistice jsou odlehlá pozorování zpravidla odebírány, jelikož představují chybná měření, anomálie atp. V tomto případě však odlišné hodnoty vyžadujeme, jelikož právě pomocí nestandardních jevů můžeme dobře kvalifikovat určité třídy.

Přehled získaných souborů a počty sloupců v plném i redukovaném souboru vyjadřuje následující tabulka.

Tabulka 9: Počet příznaků v souborech

	Redukovaná sada	**Plná sada**
Sekce	32	403
Byty	257	257
1-gramy	320	913
2-gramy	269	75138
3-gramy	218	1040613
Importované DLL	122	565
Importované funkce	1176	39565
Celkem	**2394**	**1157469**

Příznaky ve skupině *sekce* obsahují informaci o tom, jak dlouhá byla patřičná sekce v daném vzorku. V celkové sadě je dohromady 403 unikátních sekcí. Mnoho z nich je však pouze v jednom souboru. To by znamenalo jednoznačný klasifikátor, bohužel z hlediska generalizace tato informace nemá význam. Při zachování pouze odlehlých pozorování a hodnot s patřičným počtem výskytů byla vytvořena redukovaná sada o velikosti pouhých 32 příznaků.

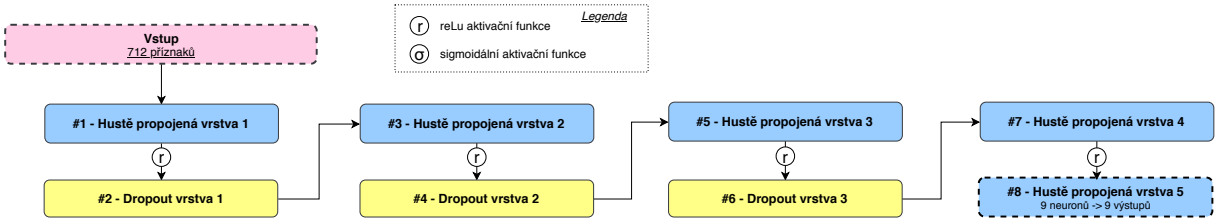
Skupina pojmenovaná *byty* nese jednoduchou informaci, která by šla vyčíst i z .bytes souborů. Celkem 257 příznaků vzniklo počítáním výskytů všech možných bytových hodnot (0 - 255) a neznámých bytů (??). Množství příznaků nebylo redukováno.

Sada n-gramů byla vytvořena čítáním výskytu importovaných a ASM funkcí. N-gramem je zde myšlena uspořádaná n-tice vyjadřující bezprostředně následující operace. Trigram (3-gram) „*db*, *nop*, *nop*“ znamenající instrukci *db*, po které následují dvě *nop* operace, se liší od jiné trojice „*nop*, *nop*, *db*“. Tyto dva konkrétní 3-gramy se mohou překrývat, což je také důvodem obrovského množství unikátních příznaků v těchto skupinách.

Poslední skupiny jsou pak tvořeny informacemi z importovacích PE tabulek. Tyto příznaky se od ostatních liší nabýváním pouze binárních hodnot vyjadřující přítomnost znaku.

6.6.1 Neuronová síť

Stejně jako v ostatních případech, i zde bylo vyzkoušeno několik variantních architektur. Síť s pouhou jednou skrytou vrstvou dokázala najít téměř optimální výsledek. Použití více vrstev přispívalo nepatrně, ale vzhledem k pouze minimálnímu nárůstu v časové náročnosti byl nakonec tento kompromis připuštěn. Jelikož však došlo k nárůstu komplexity sítě, byly rovněž použity Dropout vrstvy jako prevence proti přeučení. Výsledná architektura je znázorněna na následujícím obrázku.



Obrázek 19: Schéma architektury sítě pro zpracování assembly čítačů

Jak lze vidět v následující tabulce, nejvíce neuronů má hned první hustě propojená vrstva. Směrem k výstupní vrstvě dochází postupně ke zmenšení počtu neuronů v každé z vrstev. Tento tvar „převrácené pyramidy“ napomáhá generalizaci sítě, jelikož méně neuronů nutně znamená vyšší abstrakci prvků z předchozí vrstvy.

Tabulka 10: Finální parametry MLP sítě k analýze assembly čítačů

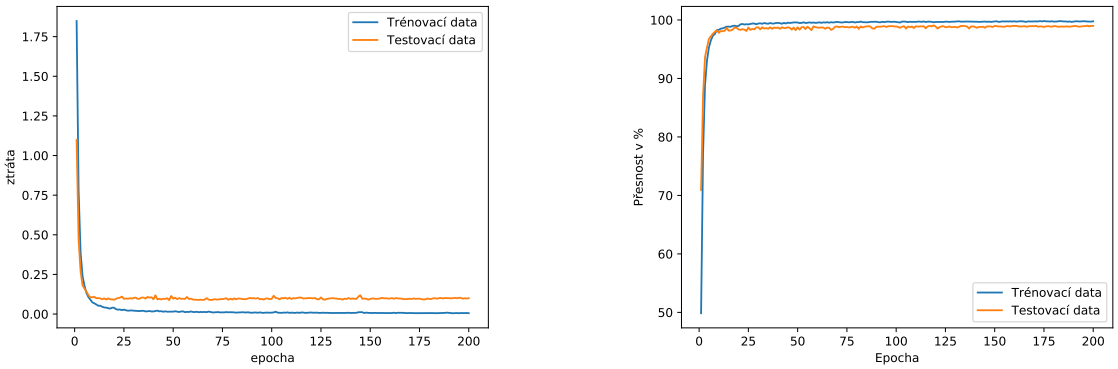
#	Název vrstvy	Parametry	Učitelné váhy	Výstup
1	Hustě propojená vr 1.	256 neuronů	182 528	256
2	Dropout vr. 1	zahodí 25%	0	256
3	Hustě propojená vr 2.	128 neuronů	32 896	128
4	Dropout vr 2.	zahodí 25%	0	128
5	Hustě propojená vr 3.	64 neuronů	8 256	64
6	Dropout vr 3.	zahodí 25%	0	64
7	Hustá propojená vr. 4.	32 neuronů	2 080	32
8	Hustě propojená vr. 5.	9 neuronů	297	9

V sekci zabývající se předzpracováním bylo zmíněno více možných vstupních množin rozdělených dle informací, které obsahují. Vyhledání optimální architektury bylo prováděno na bytovém datasetu. Dalším krokem v této analýze bylo porovnat jak se liší úspěšnosti sítí v závislosti na použitých vstupních datech a nejen z důvodu vyhledání optimální varianty, ale i pro validaci zvolené redukční operace. Výsledky tohoto zkoumání jsou zaznačeny v tabulce níže. V případě bigramů a trigramů plné sady nebylo provedeno zkoumání z důvodů časové a paměťové náročnosti.

Tabulka 11: Průměrné výsledky MLP na specifických datasetech

Dataset	**Redukovaná sada**			**Plná sada**		
	čas (s)	přesnost (%)	velikost	čas (s)	přesnost (%)	velikost
Sekce	52	95,8	32	74	96,0	403
Byty	69	96,8	257	69	96,8	257
1-gramy	74	96,6	320	91	97,9	913
2-gramy	54	97,9	269	-	-	75138
3-gramy	59	96,4	218	-	-	1040613
Importované DLL	53	92,5	122	65	92,7	565
Importované funkce	75	90,8	1176	900	91,6	39565

Ve finálním učení byla použita kombinace tří nejlepších datasetů tedy bigramy, unigramy a čítače bytů. Učení při této zvolené specifikaci bylo provedeno vícekrát, aby byla potvrzena validita experimentu. Výsledné přesnosti se lišily pouze v rámci jedné setiny procenta. Průběh učení je popsán grafy níže.

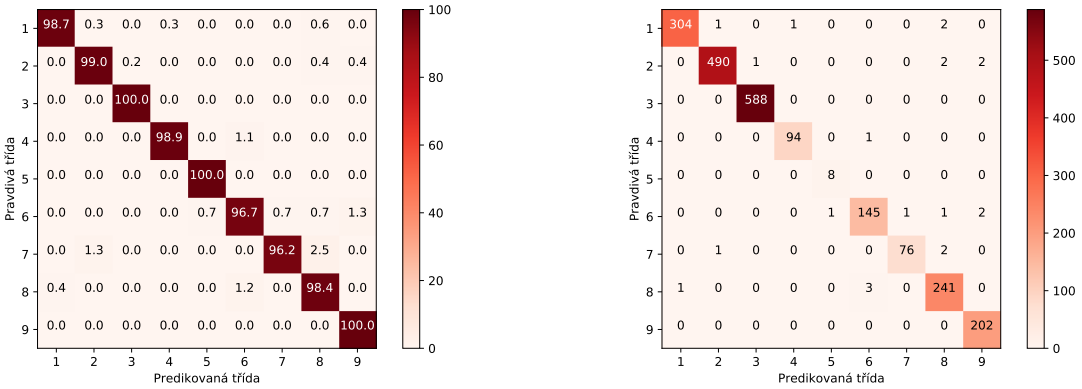


(a) Ztráta při učení

(b) Přesnost při učení

Obrázek 20: Průběh učení ASM operací gated CNN

Při zvoleném optimalizéru Adam se síť naučí klasifikovat s 97% úspěšností v prvních deseti etapách. Poté jsou již změny pouze minimální.



(a) Procentuální reprezentace

(b) Absolutní reprezentace

Obrázek 21: Konfuzní matice analýzy ASM čítačů prostřednictvím MLP

Učení celkem 200 epoch zabralo necelých 80 vteřin. Výsledkem byla síť, jež se dokázala na trénovací sadě naučit s **99,8%** úspěšností a ztrátou **0,005**, což při validaci na netrénovaných datech znamenalo přesnost **98,99%**. S tak vysokou přesností je na konfúzních maticích velmi obtížné zpozorovat jakékoliv relevantní anomálie.

6.6.2 Příbuzné klasifikátory

Jelikož tento dataset obsahuje pouze diskrétní hodnoty a vztahy mezi jednotlivými příznaky jsou prakticky neexistující. Je zvolený problém vhodný i pro vyzkoušení dalších klasifikátorů. Implementaci zajišťuje knihovna *scikit-learn*, která používá stejného rozhraní jako běžně používaný Keras, což tento experiment značně ulehčuje. V rámci této klasifikace neproběhlo žádné vyhledávání hyperparametrů, klasifikátory byly použity v jejich výchozím nastavení. Výsledky shrnuje následující tabulka.

Tabulka 12: Analýza čítačů použitím příbuzných klasifikátorů

Klasifikátor	Přesnost (%)	Doba učení (s)
Adaboost	99,5	242
Naive Bayes	91,9	5
SVC 1v1	94,1	7
Random forest	98,9	6
SVC crammer	93,2	27

Z předchozí tabulky lze vyčíst, že obecně jsou tyto klasifikátory mnohem rychlejší než samotné neuronové sítě. V případě náhodných lesů je přesnost s MLP téměř shodná, avšak za přibližně 10x menšího času. Překvapivý je pak algoritmus Adaboost z rodiny Boosting algoritmů. Bez jakéhokoliv nastavení dosáhl s přehledem nejvyšší úspěšnosti ze všech vyzkoušených analýz a klasifikátorů.

6.7 Analýza operačních sekvencí

Sekvenci instrukcí v jazyce symbolických adres, můžeme analyzovat podobně jako v případě sekvence bytů v první analýze.

Předzpracování je prakticky shodné jako zpracování *.ASM* souboru v předchozím odstavci. Podstatným rozdílem je, že výsledný soubor nebude pouze obsahovat počítadlo jednotlivých instrukcí, ale přímo kompletní sekvenci příkazů. Argumenty funkcí byly ignorovány.

Jednoduchou statistickou analýzou bylo zjištěno, že existuje celkem 913 různých operací, průměrná délka sekvence čítá přibližně 25 800 prvků, přičemž nejdelší soubor jich má téměř 500 000. Jelikož síť potřebuje vstupy stejné délky, byla experimentálně zvolena délka 75 000 operací. Soubory obsahující sekvenci o kratší délce jsou na konci doplněny vymyšlenou operací „filler“, čímž se celkový počet unikátních prvků v sekvenci zvedá na 914. Celkem 1 500 souborů je naopak ořezáno a zbylé operace jsou ignorovány.

Před učením je ještě nutné převedení textových řetězců na jejich číselnou reprezentaci. Framework Keras nabízí vlastní algoritmus pro numerickou substituci použitím hashovacích funkcí, nebo statického slovníku. Reprezentace však zůstávají zachovány pouze v rámci jednoho učení. Uložení sítě a použití tohoto naučeného modelu pro predikci používá jiný slovník, což nutně způsobuje chaotické výsledky. Perzistentní unikátnost byla zajištěna vlastním skriptem, který sekvenčně prochází všechny soubory a každé nově nalezené operaci přiřazuje unikátní inkrementující se číslo. Ve stejnou chvíli dochází k vytvoření menší varianty této mapy záměrně vynechávající funkce, které pocházejí z externího zdroje. Je možné, že tyto specifické informace nebudou mít znatelný vliv na přesnost klasifikátoru a jejich vynecháním dojde ke snížení vstupních a embedding dimenzí, čímž se nutně zvýší rychlost sítě. Při tvorbě těchto map stringů na celá čísla došlo rovněž k odstranění některých chybných prvků, které zcela očividně nebyly platnými operacemi, např. „\n, "0, "1\’“ atp.

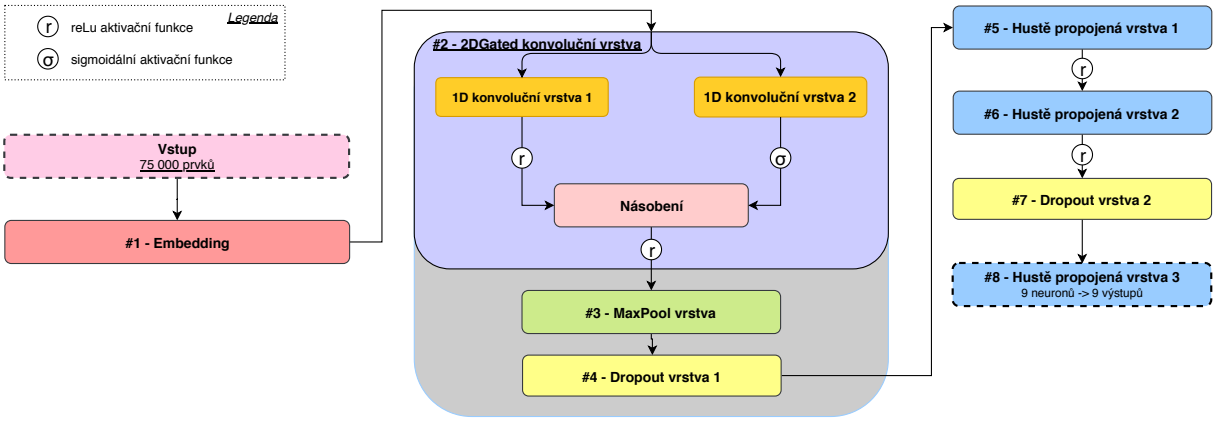
Pomocí mapovacích slovníků popsaných v předchozím odstavci, převádí finální skript všechny textové řetězce na dvě celočíselná pole. Tyto výsledky jsou následně uloženy do dvou separátních CSV souborů - redukovaného a klasického, které slouží jako vstup do samotných sítí. Tímto předzpracováním ušetříme místo na disku i dobu učení, jelikož není nutné provádět žádné další zpracování a stačí pouze číst.

K analýze těchto předzpracovaných vstupů byly postupně odzkoušeny dvě rozdílné architektury.

6.7.1 Gated CNN architektura

Tato architektura je na pohled velmi podobná s tou představenou v sekci 6.4, zaměřenou na analýzu zdrojových bytů. Jediným podstatným rozdílem je použití lokální MaxPool vrstvy s poměrně velkým kernelem. Výsledná matice je 535x větší, než-li při použití původního globálního maxima, což značně navyšuje počet trénovatelných vah první hustě propojené vrstvy. Další

změny jsou parametrického charakteru. Původní model využíval rozměrných filtrů a velkých posunů, zatímco tato varianta používá filtry mnohem menších rozměrů a jednotkový stride.



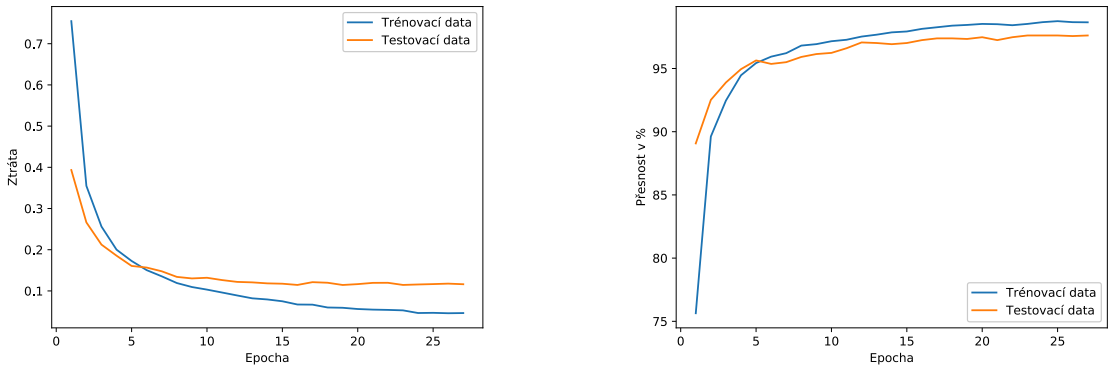
Obrázek 22: Model Gated 1D CNN sítě pro zpracování sekvence opkódů

Stejně jako v předchozích sítích i zde došlo k vyhledávání hyperparametrů. Vzhledem k delším dobám učení tyto operace probíhaly na redukovaném datasetu a pouze manuálně. Finálním optimalizátorem byl zvolen algoritmus Adagrad s učící konstantou 0,005 a o velikosti dávky pouze jednoho prvku, z důvodů paměťové náročnosti.

#	Název vrstvy	Parametry	Učitelné váhy	Výstup
1	Embedding vr.	8 dimenzí	7 280	75 000 x 8
2	Gated 1D konv. vr.	32 filtrů o délce 16	8 256	74 985 x 32
3	MaxPool vr.	128 znaků	0	585 x 32
4	Flatten vr.	-	0	18 720
5	Dropout vr. 1	zahodí 10%	0	18 720
6	Hustě propojená vr. 1	256 neuronů	4 792 576	256
7	Hustá propojená vr. 2	128 neuronů	32 896	128
8	Dropout vr. 2	zahodí 40%	0	128
9	Hustě propojená vr. 3	9 neuronů	1 161	9

Tabulka 13: Finální parametry Gated 1D CNN pro ASM

Průměrný čas pro plné vytrénování této konfigurace se pohybuje okolo jedné hodiny. Nejlepší nalezené nastavení, které je popsané v tabulce výše a jejíž učení je znázorněno grafy níže, vyžadovalo 27 epoch po 150 sekundách, což ve výsledku dává jednu hodinu a deset minut. Samotné maximum bylo nalezeno již v sedmnáctém cyklu, jelikož po následujících deset epoch nedošlo k žádnému zlepšení, síť se automaticky zastavila. Výsledkem je přesnost trénování **98,65%** a ztráta **0,0463**, což odpovídá přesnosti **97,61%** a ztrátě **0,11** na testovací množině.

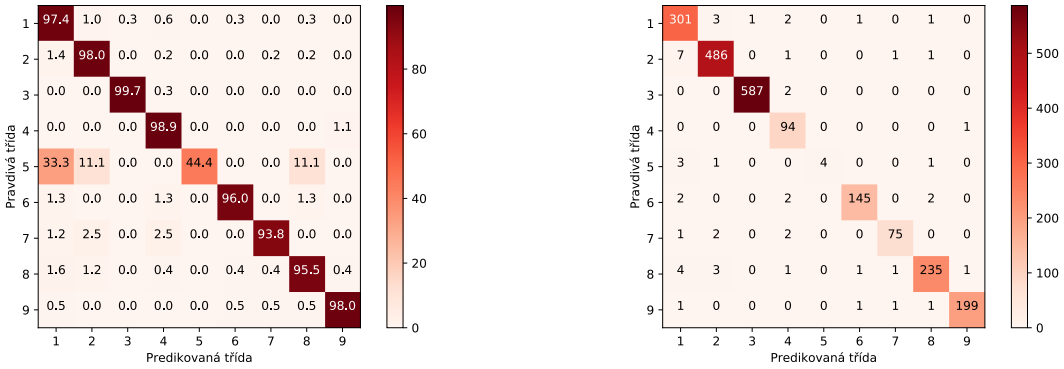


(a) Ztráta při učení

(b) Přesnost při učení

Obrázek 23: Průběh učení ASM operací gated CNN

Jak už je v tuto chvíli zvykem, nejhorší výsledky vykazuje rodina Simda s identifikačním číslem 5 z důvodu minimálního množství vzorových souborů. Druhý nejhorší výsledek má skupina reprezentující obfuskovaný malware.



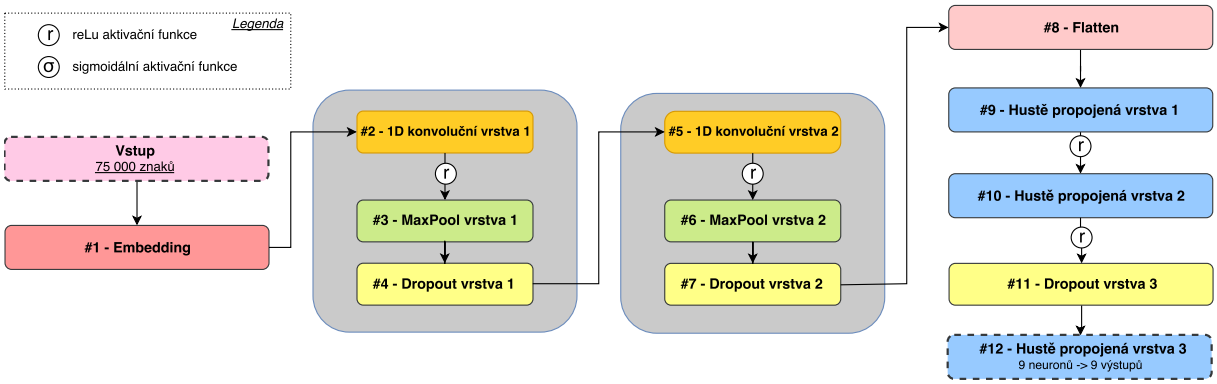
(a) Procentuální reprezentace

(b) Absolutní reprezentace

Obrázek 24: Konfuzní matice analýzy ASM operací gated CNN

6.7.2 Klasická CNN architektura

Tato konfigurace byla převzána z klasického modelu pro zpracování dvourozměrných obrázků. Jelikož jsou vstupem jedno-dimenzionální řetězce, došlo k nahrazení příslušných konvolučních a pool vrstev jejich jednodušším ekvivalentem. Ačkoliv bylo vyzkoušeno mnoho variant, nakonec se ukázalo, že původní architektura vykazuje nejlepší poměr času a přesnosti. Obrázek níže zachycuje finální podobu sítě. Jediný rozdíl, pomineme-li sníženou dimenzionalitu, je v přidané embedding vrstvě.



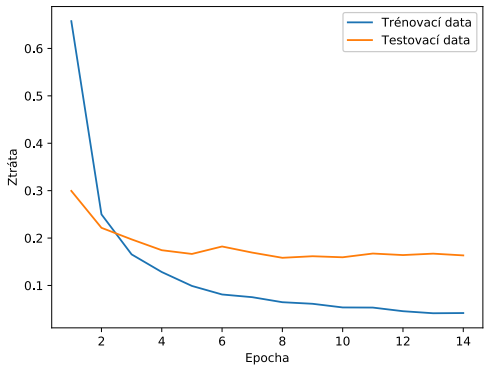
Obrázek 25: Model 1D CNN sítě pro zpracování sekvence opkódů

Podobně jako u u Gated architektury i zde byly vyzkoušeny velké filtry s velkým stridem a naopak velmi důkladné, překrývající se konvoluce. Ideální nastavení je někde uprostřed. K finálnímu testu byly zvolené filtry 32 a 64 se stridem 2 a 4. Jelikož ani zde nebyl použit globální maximální pool, vyskytuje se nejvíce vah na rozhraní mezi konvolučními a hustě propojenými vrstvami. Finální parametry spolu s váhami a rozměry výstupů popisuje následující tabulka.

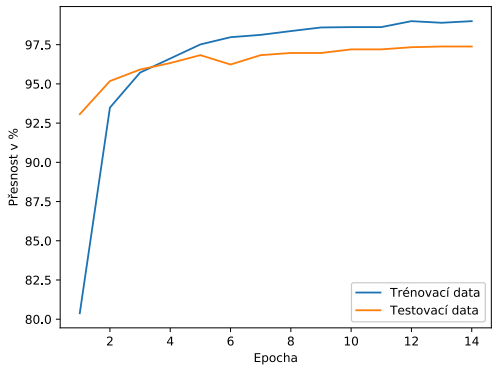
#	Název vrstvy	Parametry	Učitelné váhy	Výstup
1	Embedding vr.	8 dimenzí	7 280	75 000 x 8
2	1D konv. vr.	16 x 32, stride = 2	4 112	37 485 x 16
3	MaxPool vr.	2	0	18742 x 16
4	Dropout vr. 1	zahodí 10%	0	18742 x 16
5	1D konv. vr.	32 x 64, stride = 4	32 800	4670 x 32
6	MaxPool vr.	4	0	1 167 x 32
7	Dropout vr. 2	zahodí 10%	0	1 167 x 32
8	Flatten vr.	-	0	37 344
9	Hustě propojená vr. 1	256 neuronů	9 560 320	256
10	Hustá propojená vr. 2	128 neuronů	32 896	128
11	Dropout vr. 2	zahodí 50%	0	128
12	Hustě propojená vr. 3	9 neuronů	1 161	9

Tabulka 14: Finální parametry klasickou 1D CNN pro ASM

Oproti předchozím analýzám lze na grafu tohoto učení zpozorovat poměrně velké rozdíly mezi trénovací a testovací sadou. Celkově je proces učení velmi nevýrazný. Ve výsledku se však síť zastavila mechanismem early stopping již ve 14 epoše po 42,5 minutách. Výsledná přesnost na množině validačních dat se v této epoše rovnala **97,38 %**.



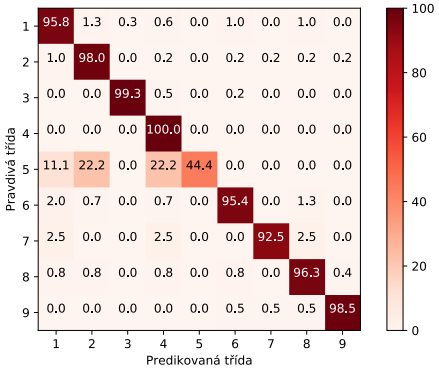
(a) Ztráta při učení



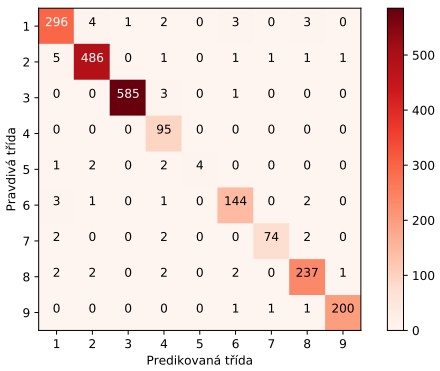
(b) Přesnost při učení

Obrázek 26: Průběh učení ASM operací klasickou CNN

Při pohledu na konfuzní matici níže, lze zpozorovat nejkritičtější chyby v klasifikaci. Zcela očekávaně je problém v rozpoznání 5. třídy, která obsahuje pouhých 42 vzorků, přičemž k učení jich bylo použito pouze 35. Další problematickou třídou je 8. rodina, což je skupina, na které byla použita obfuskace. Tyto chyby jsou odůvodnitelné ztrátou informace o špatně přeložených bytech, jež byla zmíněna v sekci o předzpracování dat.



(a) Procentuální reprezentace



(b) Absolutní reprezentace

Obrázek 27: Konfuzní matice analýzy ASM operací klasickou CNN

6.8 Souhrn

Následující tabulka předhledně zobrazuje nejlepší nalezené výsledky napříč všemi analýzami.

Vstup	Klasifikátor	Přesnost (%)	Čas (m)
Sekvence bytů	Gated CNN	98,40	432,0
Bytový obraz	2D CNN	97,12	34,0
Čítač ASM příznaků	MLP	98,99	1,2
Čítač ASM příznaků	AdaBoost	99,50	4,0
Sekvence operací	Gated CNN	97,61	70,0
Sekvence operací	1D CNN	97,38	42,5

Tabulka 15: Srovnání nejlepších experimentů

Ačkoliv se jeví analýza čítačů jako nejlepší a zároveň nejrychlejší, je nutno brát v potaz, že tyto vstupní data byly vybrány na míru přesně pro tyto třídy a předcházelo jim zdlouhavé předzpracování všech vzorků. Při reálném použití v antivirových nástrojích, kdy by možných tříd bylo mnohonásobně vyšší množství, se dá očekávat nižší úspěšnost.

Další skvělé výsledky měla překvapivě metoda „prohledávající“ celou nepozměněnou binární reprezentaci souboru. Tímto tedy odpadá nutnost jakéhokoliv předzpracování, či znalostí vnitřní struktury skenovaného souboru. Rovněž je možné tento přístup použít na více typů souborů. Časová nevýhoda je pak mírně negována mnohem výkonějšími stroji, kterými společnostmi provádějící tyto analýzy zpravidla disponují.

Analýza operačních sekvencí je v zásadě shodná s analýzou klasické bytové reprezentace. Jediným rozdílem je nutnost předzpracování. Na rozdíl od předchozí sekvence je zde nutné provézt proces dissassembly.

Analýza zmenšené dvou-dimenzionální reprezentace měla nejhorší výsledky z testovaných metod, ale i tak byla poměrně úspěšnou. Stejně jako v případě analýzy čítačů zde nedochází k učení na chování vzorků, ale na jejich struktuře a vizuální, či statistické podobnosti.

6.9 Neprozkoumané analýzy

Výše uvedené analýzy prozkoumaly pouze malou část možných vstupních dat. Cílem této sekce je stručně popsat další možné úhly pohledu.

Zvolený dataset obsahoval pouze dissassemblovaná data, která byla navíc zbavena PE hlavičky. V případě analýzy bitových sekvencí jsme tedy přišli o data, která ve výsledku mohla přispět k lepším výsledkům. Díky absenci PE hlavičky nemohou být tyto vzorky spuštěny, respektive nemůže být provedena dynamická analýza. Prostřednictvím této analýzy bychom mohli získat sekvence prováděných funkcí podobně jako v případě analýzy operačních sekvencí. Lze se tedy domnívat, že stejné modely, které fungovaly v případě assembly funkcí, budou použitelné i zde. Pokud nebyla použita žádná obrana proti dynamické analýze, měla by tato metoda vynést mnohem lepší výsledky než varianta statická.

V případě assembly čítačů byly nejprve vyzkoušeny jednotlivé sady vstupních příznaků samostatně. Už v tomto případě dosahovaly experimenty přívětivých výsledků. Avšak v případě

spojení více jednotlivých sad dohromady, se síť dokázala naučit ještě o něco více. Podobný princip by mohl být použit i v případě celých naučených sítí. Každý tento klasifikátor používá odlišné vstupní informace a klasifikuje třídy s různou úspěšností. Nabízí se tedy možnost tyto vstupy také klasifikovat v rámci jednoho celku. Existuje více způsobů provedení. Nejjednodušší způsob by byl využití již naučených sítí, kdy by postupně došlo k predikci třídy každého vzorku všemi sítěmi. Z výsledných odpovědí by se pak vzala skupina s největším počtem „hlasů“. Druhým způsob je možný díky frameworku Keras. Ten umožňuje editaci naučených sítí a rovněž dovoluje spojit více struktur dohromady. Je tedy možné zcela vypustit poslední hustě propojenou vrstvu neurálních sítí a místo toho všechny vrstvy napojit na jedinou spojovací vrstvu. Poté by bylo potřeba všechny nově vzniklé vrstvy naučit. Poslední možností by bylo zkonstruovat jednu komplikovanou síť s více vstupy. Učení takovéto sítě by však na stávající platformě trvalo v řádu dní.

Podobně by pak šlo pokračovat i s případnou dynamickou analýzou. Spojení dynamických a statických postupů bývá často označováno jako hybridní analýza.

Dalším možným zdrojem informací běžně užívaným v klasické statické analýze je entropie. Jedná se o podíl „neurčitosti“ v daném zdroji. Jinými slovy nulová entropie bude náležet souboru obsahujícímu pouze jeden konstantní repetitivní znak. Naopak vysoká entropie je způsobena souborem s plně náhodnými informacemi. Prakticky se entropie používá k detekci zašifrovaných a zkompresovaných souborů a sekcí, případně naopak lze ji použít k vyhledání dekódovací funkce v plně zašifrovaném programu.

7 Závěr

V rámci této diplomové práce došlo k představení problematiky malware a důvodu proč je důležité použít metodu strojového učení k jejich rozpoznání a klasifikaci. Dále byly popsány principy nejdůležitějších částí neuronových sítí. Na základě těchto informací došlo ke zpracování vybraného datasetu do čtyř různých vstupních sad, pomocí kterých byla provedena klasifikace neuronovými sítěmi a příbuznými metodami. Přesnost žádného z použitých klasifikátorů neklesla pod 97%, lze tedy tvrdit, že všechny experimenty lze pokládat za úspěšné.

Nejlepších výsledků dosahovala metoda, která počítala výskyt různých znaků v každém ze souborů. Tato metoda je však založena spíše na statických metodách, než-li naučení chování. To ve výsledku znamená menší využitelnost z hlediska klasifikace malware dle jejich typů (spyware, vir, červ atp.). Podobnou nevýhodu má teoreticky i analýza bytového obrazu. Zbývající dvě metody analyzující sekvenci bytů, či disassemblovaných funkcí, by teoreticky šly využít i k mnohem obecnější klasifikaci. U analýz sekvencí je však nutno počítat se značně delšími časy učení.

Získané výsledky nemusí být zcela nejlepšími. Dlouhotrvající učení značně znepříjemnilo optimalizaci. V případě lepší HW platformy by bylo možné použít důkladnější analýzy a důslednější optimalizace. Největší problém zapříčinil starší plotnový disk a nedostatečná kapacita pamětí.

Uvedené metody tvoří pouze zlomek možných experimentů. Pouhým spojením některých klasifikátorů do jednoho celku by mohlo dojít k získání lepších výsledků na úkor větší časové náročnosti. Na konci experimentální části bylo rovněž popsáno několik dalších postupů, které by mohly přinést další zisk v přesnosti.

Literatura

[1] MINIHANE, Niamh, Francisca MORENO, Eric PETERSON, Raj SAMANI, Craig SCHMUGAR, Dan SOMMER a Bing SUN. *McAfee Labs Threat Report: December 2017* [online]. McAfee, 2017 [cit. 2018-03-02].

[2] Malware naming convention. *Microsoft: Windows Defender Security Intelligence* [online]. Redmond: Microsoft Corporation, 2018 [cit. 2018-03-03]. Dostupné z: https://www.microsoft.com/en-us/wdsi/help/malware-naming

[3] *Security Report 2016/2017* [online]. Magdeburg: AV-TEST, 2017 [cit. 2018-04-10]. Dostupné z: https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2016-2017.pdf

[4] SZOR, Peter. *Počítačové viry: analýza útoku a obrana*. Přeložil Lukáš PELIKÁN, přeložil Roman SKŘIVÁNEK. Brno: Zoner Press, 2006. ISBN 80-86815-04-8.

[5] RAMACHANDRAN, Rajit, Barret ZOPH a Quoc V. LE. *Searching for Activation Functions* [online]. [cit. 2018-04-27]. Dostupné z: https://arxiv.org/abs/1710.05941

[6] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. *Deep learning*. Cambridge, Massachusetts: The MIT Press, 2016. ISBN 978-0262035613.

[7] HAYKIN, Simon S. *Neural networks and learning machines*. 3rd ed. New York: Prentice Hall, c2009. ISBN 978-0-13-147139-9.

[8] Embeddings. *TensorFlow* [online]. 2018, 30. březen 2018 [cit. 2018-04-27]. Dostupné z: https://www.tensorflow.org/versions/master/programmers_guide/embedding

[9] MIKOLOV, Tomas, Kai CHEN, Greg CORRADO a Jeffrey DEAN. *Efficient Estimation of Word Representations in Vector Space* [online]. 2013 [cit. 2018-04-07]. Dostupné z: https://arxiv.org/pdf/1301.3781.pdf

[10] PENNINGTON, Jeffrey, Richard SOCHER a Christopher D. MANNING. *GloVe: Global Vectors for Word Representation* [online]. Stanford, 2014 [cit. 2018-04-07]. Dostupné z: https://nlp.stanford.edu/pubs/glove.pdf. Stanford University.

[11] SRISTAVA, Nitish, Geoffrey HINTON, Alex KRIZHEVSKY, Ilya SUTSKEVER a Ruslan SALAKHUTDINOV. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* [online]. 2014, **2014**(15), 30 [cit. 2018-04-07]. Dostupné z: https://www.cs.toronto.edu/ hinton/absps/JMLRdropout.pdf

[12] RUDER, Sebastian. *An overview of gradient descent optimization algorithms* [online]. Dublin, 2017 [cit. 2018-04-27]. Dostupné z: https://arxiv.org/pdf/1609.04747.pdf

[13] RIOS, Luis Miguel a Nikolaos V. SAHINIDIS. *Derivative-free optimization: A review of algorithms and comparison of software implementations* [online]. Pittsburgh [cit. 2018-04-27]. Dostupné z: http://thales.cheme.cmu.edu/dfo/papers/dfo.pdf

[14] BENGIO, Yoshua, Jérôme LOURADOUR, Ronan COLLOBERT a Jason WESTON. *Curriculum Learning* [online]. 2009 [cit. 2018-04-27]. Dostupné z: https://ronan.collobert.com/pub/matos/2009_curriculum_icml.pdf

[15] IOFFE, Sergey a Christian SZEGEDY. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift* [online]. 2015 [cit. 2018-04-27]. Dostupné z: https://arxiv.org/pdf/1502.03167.pdf

[16] LECUN, Yann, Léon BOTTOU, Yoshua BENGIO a Patrick HAFFNER. *Gradient Based Learning Applied to Document Recognition* [online]. 1998 [cit. 2018-03-02]. Dostupné z: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

[17] DAUPHIN, Yann N., Angela FAN, Michael AULI a David GRANGIER. *Language Modeling with Gated Convolutional Networks* [online]. 2017 [cit. 2018-03-02]. Dostupné z: https://arxiv.org/pdf/1612.08083.pdf

[18] GIDUDU, Anthony, Gregg HULLEY a Tshilidzi MARWALA. *Image Classification Using SVMs: One-against-One Vs One-against-All* [online]. Witwatersrand [cit. 2018-03-02].

[19] ED.: ROSNI ABDULLAH ... *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2011 Sixth International Conference on*. Piscataway, NJ: IEEE, 2011. ISBN 9781457710926.

[20] KARTHIKEYAN, Shanmugavadivel, Grégoire JACOB, Shanmugavadivel NATARAJ a Lakshmanan NATARAJ. *Malware Images: Visualization and Automatic Classification* [online]. 2011 [cit. 2018-03-02]. Dostupné z: https://www.researchgate.net/publication/228811247_Malware_Images_Visualization_and_Automatic_Classification

[21] ANJU, S. S., P. HARMYA, N. JAGADEESH a R. DARSANA. *Malware detection using assembly code and control flow graph optimization* [online]. Coimbatore, 2010 [cit. 2018-04-27]. Dostupné z: https://dl.acm.org/citation.cfm?id=1858443&dl=ACM&coll=DL

[22] DAM, Khanh-Huu-The a Tayssir TOUILI. *Malware Detection based on Graph Classification* [online]. Villetaneuse, 2017 [cit. 2018-04-27]. Dostupné z: http://www.scitepress.org/Papers/2017/62095/62095.pdf

[23] RAFF, Edward, Jon BARKER, Jared SYLVESTER, Robert BRANDON, Bryan CATANZARO a Charles NICHOLAS. *Malware Detection by Eating a Whole EXE* [online]. 2017 [cit. 2018-04-07].

[24] JAENISCH, Holger M., Andrew N. POTTER, Deborah WILLIAMS a James W. HANDLEY. *Fractals, Malware, and Data Models* [online]. 2015 [cit. 2018-04-27]. Dostupné z: https://www.researchgate.net/publication/258716923_Fractals_malware_and_data_models

[25] RONEN, Royi, Marian RADU, Corina FEUERSTEIN, Elad YOM-TOV a Mansour AHMADI. Microsoft Malware Classification Challenge [online]. 2018 [cit. 2018-03-02]. Dostupné z: https://arxiv.org/pdf/1802.10135

[26] THOMPSON, Steven K. *Sampling*. 3rd ed. Hoboken, N.J.: Wiley, c2012, s. 139-156. Wiley series in probability and statistics. ISBN 9780470402313.

[27] CIMPANU, Catalin. Ramnit Botnet Comeback Continues in 2017. *Bleeping Computer* [online]. 2017, 20.02.2017 [cit. 2018-04-27]. Dostupné z: https://www.bleepingcomputer.com/news/security/ramnit-botnet-comeback-continues-in-2017/

[28] W32.Ramnit. *Symantec* [online]. [cit. 2018-04-27]. Dostupné z: https://www.symantec.com/security-center/writeup/2010-011922-2056-99

[29] Adware:Win32/Lollipop. *Microsoft: Windows Defender Security Intelligence* [online]. Redmond: Microsoft Corporation, 2013 [cit. 2018-04-27]. Dostupné z: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Adware:Win32/Lollipop

[30] The Kelihos Botnet. *Malwaretech* [online]. 2017 [cit. 2018-04-27]. Dostupné z: https://www.malwaretech.com/2017/04/the-kelihos-botnet.html

[31] Trojan.Vundo. *Symantec* [online]. 2004, 20.11.2004 [cit. 2018-04-27]. Dostupné z: https://www.symantec.com/security-center/writeup/2004-112111-3912-99

[32] Win32/Simda. *Microsoft: Windows Defender Security Intelligence* [online]. 2013, 05.09.2013 [cit. 2018-04-27]. Dostupné z: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Simda

[33] Win32/Tracur. *Microsoft: Windows Defender Security Intelligence* [online]. 2011, 30.06.2011 [cit. 2018-04-27]. Dostupné z: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FTracur

[34] Gatak: Healthcare organizations in the crosshairs. Symantec Official Blog [online]. 2016, 21.11.2016 [cit. 2018-04-27]. Dostupné z: https://www.symantec.com/connect/blogs/gatak-healthcare-organizations-crosshairs

[35] IDA: About. *Hex-rays* [online]. 2015, 27.03.2015 [cit. 2018-04-27]. Dostupné z: https://www.hex-rays.com/products/ida/

[36] MILLION, Elizabeth. *The Hadamard Product* [online]. Tacoma, 2007 [cit. 2018-04-27]. Dostupné z: http://buzzard.ups.edu/courses/2007spring/projects/million-paper.pdf

[37] LIN, Min, Qiang CHEN a Shuicheng YAN. *Network in Network* [online]. Singapore, 2014 [cit. 2018-04-06]. Dostupné z: https://arxiv.org/pdf/1312.4400.pdf. National University of Singapore, Singapore.

[38] LYSENKO, Oleksandr. *Kaggle-malware-classification/pe_parser.py* [online]. GitHub, 2015 [cit. 2018-04-26]. Dostupné z: https://github.com/sash-ko/kaggle-malware-classification/blob/master/pe_parser.py

[39] HAHN, Katja. *Robust Static Analysis of Portable Executable Malware* [online]. Leipzig, 2007 [cit. 2018-04-27]. Dostupné z: http://buzzard.ups.edu/courses/2007spring/projects/million-paper.pdf. Master Thesis in Computer Science. HTWK Leipzig.

A Obsah CD

Kód pro vypracování analýz zvoleného datasetu se nachází na přiloženém fyzickém médiu. Nejedná se o ucelený program, nýbrž o sadu mnoha samostatných skriptů. Tato příloha popisuje funkcionalitu jednotlivých souborů a vzorové použití. V nezměněné podobě nebude většina přiložených skriptů funkční, jelikož samotný dataset nemohl být k této práci přiložen z důvodu jeho rozměrů. Stále je k dispozici ke stažení na originálním umístění soutěže[2]. Skripty nepracují s archivy - vyžadují dekomprimované soubory, jež mají celkovou velikost asi 250 GB.

V rámci této diplomové práce bylo vytvořeno přibližně 50 skriptů, přičemž 37 z nich bylo použito k vypracování současné podoby tohoto textu. Tyto vybrané skripty budou v této podsekci popsány v abecedním pořadí.

Dataset.py Třída reprezentující celkový dataset

helper_asm.py Funkce pro parsování .ASM souboru.

helper_dictionary_for_1DASM.py Tento soubor byl použit pro přečtení souborů obsahujících sekvence operací za účelem získání všech možných unikátních prvků v řetězci. Každému unikátu byl přiřazen celočíselný identifikátor. Výsledný slovník byl uložen staticky do tohoto souboru, aby nemuselo docházet k opětovnému spuštění při případném opětovném předzpracování.

helper_plot.py Pomocné funkce k vykreslování grafů.

NET_*.py Třídy reprezentující neuronové sítě a jejich učící a validační mechanismy.

obfuscate_base64.py Skript použitý pro ukázku byte64 obfuskace se substitucí (výpis 2).

plot_activations.py Skript použitý pro ukázku grafů aktivačních funkcí v tabulce 1.

plot_from_2_csv_comparison.py Poskytuje srovnání výkonnosti dvou sítí.

plot_from_csv.py Vytvoření grafů průběhu učení sítí ze souboru CSV.

preproc_asm_computed.py Získává počty bytových hodnot z ASM souboru.

preproc_asm_statistics.py Účelem tohoto skriptu je vypracování statistiky různých příznaků dle kategorií v datasetu. Pro každý příznak je vypracováno v kolika procentech souborů se vyskytuje, případně kolik těchto příznaků je celkem ve všech souborech jedné rodiny. Tato statistika je důležitá pro redukci počtu příznaků.

preproc_asm_statistics_shorter.py Zkrácená verze předchozího skriptu, zabývající se pouze ngramy.

[2] `https://www.kaggle.com/c/malware-classification/data`

preproc_colored_binary.py description

preproc_convert_set_to_int.py Převádí CSV soubor obsahující operace z .ASM souboru na jiný CSV soubor skládající se pouze z celočíselných hodnot.

preproc_image_binary.py Převod binárního kódu do obrazové formy.

preproc_image_colored.py Stejně jako předchozí skript, ale obarvený - nevhodný pro strojové čtení, ale více vypovídající pro lidského pozorovatele.

preproc_image_flow.py Rozmístění jednotlivých obrázků do složek pro rychlé čtení knihovnou Keras.

preproc_image_rescaler.py Zmenšení obrázků s dynamickou velikostí na fixní čtvercové dimenze, které jsou nutné pro dvou-dimenzionální analýzu.

preproc_mks_binary.py Převedení originálního byte souboru do celočíselné podoby bez adres.

preproc_msg_binary.py Stejně jako předchozí skript, ale navíc probíhá komprese.

Provider_*.py Skupina tříd, které poskytují vzorky pro dané modely v paralelizovatelné formě.

Sample.py Třída reprezentující konkrétní vzorek - cestu a kategorii.

start_class_counter.py Provedení klasifikace pomocí „příbuzných metod“ k neuronovým sítím.

start_csv_normalisation.py Skript pro normalizaci získaných CSV čítačů použitím L2 normalizace.

start_feature_extractor*.py Spouštěcí skript pro parsování ASM souboru a získání příslušných čítačů.

start_net_*.py Spouštěcí soubory pro jednotlivé analýzy.

start_reduce_sets.py Redukce nepotřebných příznaků na základě detekce odlehlých pozorování.

static_features.py Jednoduchý statický soubor obsahující seznam všech příznaků. Slouží převážně k testování a statistice.

validate_model_*.py Některé experimenty selhaly vlivem špatného zpracování výsledků. Tento skript sloužil k otevření uložené neuronové sítě a opětovného provedení validace.